LwXMLP information	Value
Version	1.0.0

MISRA Version	MISRA C:2012

Violations	Total
Required Directives	0
Required Rules	**2**
Advisory Directives	0
Advisory Rules	**1**

Directive/ Rule	Category	Rule	Rational	PC-Lint Result
The implementation				
D.1.1	Required	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood		Pass
Compilation and build				
D.2.1	Required	All source files shall compile without any compilation errors		Pass
Requirements traceability				
D.3.1	Required	All code shall be traceable to documented requirements		Pass
Code design				
D.4.1	Required	Run-time failures shall be minimized		Pass
D.4.2	Required	All usage of assembly language should be documented		Pass
D.4.3	Required	Assembly language shall be encapsulated and isolated		Pass
D.4.4	Required	Sections of code should not be commented out		Pass
D.4.5	Advisory	Identifiers in the same namespace with overlapping visibility should be typographically unambiguous		Pass
D.4.6	Advisory	typedefs that indicate size and signedness should be used in place of the basic numerical types		Pass
D.4.7	Required	If a function returns error information, then that error information shall be tested		Pass
D.4.8	Advisory	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden		Pass
D.4.9	Advisory	A function should be used in preference to a function-like macro where they are interchangeable		Pass
D.4.10	Required	Precautions shall be taken in order to prevent the contents of a header file being included more than once		Pass
D.4.11	Required	The validity of values passed to library functions shall be checked		Pass
D.4.12	Required	Dynamic memory allocation shall not be used	the dynamic allocation is supported and the static allocation as well. if the rule is checked while the static allocation is supported the rule will pass	Fail
D.4.13	Advisory	Functions which are designed to provide operations on a resource should be called in an appropriate sequence		Pass
A standard C environment				
R.1.1	Required	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits		Pass
R.1.2	Advisory	Language extensions should not be used		Pass
R.1.3	Required	There shall be no occurrence of undefined or critical unspecified behaviour		Pass
Unused Code				
R.2.1	Required	A project shall not contain unreachable code		Pass

R.2.2	Required	There shall be no dead code		Pass
R.2.3	Advisory	A project should not contain unused type declarations		Pass
R.2.4	Advisory	A project should not contain unused tag declarations	the parser instance is defined as struct in the LwXMLP_PRIVATE_TYPES.h to allow the instance to be public in case of external instance allocation while provate when internal instance allocation	Fail
R.2.5	Advisory	A project should not contain unused macro declarations		Pass
R.2.6	Advisory	A function should not contain unused label declarations		Pass
R.2.7	Advisory	There should be no unused parameters in functions		Pass
Comments				
R.3.1	Required	The character sequences /* and // shall not be used within a comment		Pass
R.3.2	Required	Line-splicing shall not be used in // comments		Pass
Character sets lexical conventions				
R.4.1	Required	Octal and hexadecimal escape sequences shall be terminated		Pass
R.4.2	Advisory	Trigraphs should not be used		Pass
Identifiers				
R.5.1	Required	External identifiers shall be distinct		Pass
R.5.2	Required	Identifiers declared in the same scope and name space shall be distinct		Pass
R.5.3	Required	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope		Pass
R.5.4	Required	Macro identifiers shall be distinct		Pass
R.5.5	Required	Precautions shall be taken in order to prevent the contents of a header file being included more than once		Pass
R.5.6	Required	A typedef name shall be a unique identifier		Pass
R.5.7	Required	A tag name shall be a unique identifier		Pass
R.5.8	Required	Identifiers that define objects or functions with external linkage shall be unique		Pass
R.5.9	Advisory	Identifiers that define objects or functions with internal linkage should be unique		Pass
Types				
R.6.1	Required	Bit-fields shall only be declared with an appropriate type		Pass
R.6.2	Required	Single-bit named bit fields shall not be of a signed type		Pass
Literals and constants				
R.7.1	Required	Octal constants shall not be used		Pass
R.7.2	Required	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type		Pass
R.7.3	Required	The lowercase character 'l' shall not be used in a literal suffix		Pass
R.7.4	Required	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"		Pass
Declarations and definitions				
R.8.1	Required	Types shall be explicitly specified		Pass
R.8.2	Required	Function types shall be in prototype form with named parameters		Pass
R.8.3	Required	All declarations of an object or function shall use the same names and type qualifiers		Pass
R.8.4	Required	A compatible declaration shall be visible when an object or function with external linkage is defined		Pass
R.8.5	Required	An external object or function shall be declared once in one and only one file		Pass
R.8.6	Required	An identifier with external linkage shall have exactly one external definition		Pass

R.8.7	Advisory	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit		Pass
R.8.8	Required	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage		Pass
R.8.9	Advisory	An object should be defined at block scope if its identifier only appears in a single function		Pass
R.8.10	Required	An inline function shall be declared with the static storage class		Pass
R.8.11	Advisory	When an array with external linkage is declared, its size should be explicitly specified		Pass
R.8.12	Required	When an array with external linkage is declared, its size should be explicitly specified		Pass
R.8.13	Advisory	A pointer should point to a const-qualified type whenever possible.		Pass
R.8.14	Required	The restrict type qualifier shall not be used		Pass
Initialization				
R.9.1	Mandatory	The value of an object with automatic storage duration shall not be read before it has been set		Pass
R.9.2	Required	The initializer for an aggregate or union shall be enclosed in braces		Pass
R.9.3	Required	Arrays shall not be partially initialized		Pass
R.9.4	Required	An element of an object shall not be initialised more than once		Pass
R.9.5	Required	Where designated initialisers are used to initialize an array object the size of the array shall be specified explicitly		Pass
The essential type model				
R.10.1	Required	Operands shall not be of an inappropriate essential type		Pass
R.10.2	Required	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations		Pass
R.10.3	Required	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category		Pass
R.10.4	Required	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category		Pass
R.10.5	Advisory	The value of an expression should not be cast to an inappropriate essential type		Pass
R.10.6	Required	The value of a composite expression shall not be assigned to an object with wider essential type		Pass
R.10.7	Required	If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type		Pass
R.10.8	Required	The value of a composite expression shall not be cast to a different essential type category or a wider essential type		Pass
Pointers type conversions				
R.11.1	Required	Conversions shall not be performed between a pointer to a function and any other type		Pass
R.11.2	Required	Conversions shall not be performed between a pointer to incomplete and any other type		Pass
R.11.3	Required	A cast shall not be performed between a pointer to object type and a pointer to a different object type		Pass
R.11.4	Advisory	A conversion should not be performed between a pointer to object and an integer type		Pass

R.11.5	Advisory	A conversion should not be performed from pointer to void into pointer to object		Pass
R.11.6	Required	A cast shall not be performed between pointer to void and an arithmetic type		Pass
R.11.7	Required	A cast shall not be performed between pointer to object and a non-integer arithmetic type		Pass
R.11.8	Required	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer		Pass
R.11.9	Required	The macro NULL shall be the only permitted form of integer null pointer constant		Pass
Expressions				
R.12.1	Advisory	The precedence of operators within expressions should be made explicit		Pass
R.12.2	Required	The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand		Pass
R.12.3	Required	The comma operator should not be used		Pass
R.12.4	Required	Evaluation of constant expressions should not lead to unsigned integer wrap-around		Pass
Side effects				
R.13.1	Required	Initialiser lists shall not contain persistent side-effects		Pass
R.13.2	Required	The value of an expression and its persistent side-effects shall be the same under all permitted evaluation orders		Pass
R.13.3	Advisory	A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator		Pass
R.13.4	Advisory	The result of an assignment operator should not be used		Pass
R.13.5	Required	The right hand operand of a logical && or || operator shall not contain persistent side-effects		Pass
R.13.6	Mandatory	The operand of the sizeof operator shall not contain any expression which has potential side-effects		Pass
Control statement expressions				
R.14.1	Required	A loop counter shall not have essentially floating type		Pass
R.14.2	Required	A for loop shall be well-formed		Pass
R.14.3	Required	Controlling expressions shall not be invariant		Pass
R.14.4	Required	The controlling expression of an if-statement and the controlling expression of an iteration-statement shall have essentially Boolean type		Pass
Control flow				
R.15.1	Advisory	The goto statement should not be used		Pass
R.15.2	Required	The goto statement shall jump to a label declared later in the same function		Pass
R.15.3	Required	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement		Pass
R.15.4	Advisory	There should be no more than one break or goto statement used to terminate any iteration statement		Pass
R.15.5	Advisory	A function should have a single point of exit at the end		Pass
R.15.6	Required	The body of an iteration-statement or a selection-statement shall be a compound statement		Pass
R.15.7	Required	All if ... else if constructs shall be terminated with an else statement		Pass
Switch statements				
R.16.1	Required	All switch statements shall be well-formed		Pass

R.16.2	Required	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement		Pass
R.16.3	Required	An unconditional break statement shall terminate every switch-clause		Pass
R.16.4	Required	Every switch statement shall have a default label		Pass
R.16.5	Required	A default label shall appear as either the first or the last switch label of a switch statement		Pass
R.16.6	Required	Every switch statement shall have at least two switch-clauses		Pass
R.16.7	Required	A switch-expression shall not have essentially Boolean type		Pass
Functions				
R.17.1	Required	The features of <stdarg.h> shall not be used		Pass
R.17.2	Required	Functions shall not call themselves, either directly or indirectly		Pass
R.17.3	Mandatory	A function shall not be declared implicitly		Pass
R.17.4	Mandatory	All exit paths from a function with non-void return type shall have an explicit return statement with an expression		Pass
R.17.5	Advisory	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements		Pass
R.17.6	Mandatory	The declaration of an array parameter shall not contain the static keyword between the [ ]		Pass
R.17.7	Required	The value returned by a function having non-void return type shall be used		Pass
R.17.8	Advisory	A function parameter should not be modified		Pass
Pointers and arrays				
R.18.1	Required	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand		Pass
R.18.2	Required	Subtraction between pointers shall only be applied to pointers that address elements of the same array		Pass
R.18.3	Required	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object		Pass
R.18.4	Advisory	The +, -, += and -= operators should not be applied to an expression of pointer type		Pass
R.18.5	Advisory	Declarations should contain no more than two levels of pointer nesting		Pass
R.18.6	Required	The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist		Pass
R.18.7	Required	Flexible array members shall not be declared		Pass
R.18.8	Required	Variable-length array types shall not be used		Pass
Overlapping storage				
R.19.1	Mandatory	An object shall not be assigned or copied to an overlapping object		Pass
R.19.2	Advisory	The union keyword should not be used		Pass
Preprocessing directives				
R.20.1	Advisory	#include directives should only be preceded by preprocessor directives or comments		Pass

R.20.2	Required	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name		Pass
R.20.3	Required	The #include directive shall be followed by either a <filename> or "filename" sequence		Pass
R.20.4	Required	A macro shall not be defined with the same name as a keyword		Pass
R.20.5	Advisory	#undef should not be used		Pass
R.20.6	Required	Tokens that look like a preprocessing directive shall not occur within a macro argument		Pass
R.20.7	Required	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses		Pass
R.20.8	Required	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1		Pass
R.20.9	Required	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation		Pass
R.20.1	Advisory	The # and ## preprocessor operators should not be used		Pass
R.20.11	Required	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator		Pass
R.20.12	Required	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators		Pass
R.20.13	Required	A line whose first token is # shall be a valid preprocessing directive		Pass
R.20.14	Required	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related		Pass
Standard Libraries				
R.21.1	Required	#define and #undef shall not be used on a reserved identifier or reserved macro name		Pass
R.21.2	Required	A reserved identifier or macro name shall not be declared		Pass
R.21.3	Required	The memory allocation and deallocation functions of <stdlib.h> shall not be used	the dynamic allocation is supported and the static allocation as well. if the rule is checked while the static allocation is supported the rule will pass	Fail
R.21.4	Required	The standard header file <setjmp.h> shall not be used		Pass
R.21.5	Required	The standard header file <signal.h> shall not be used		Pass
R.21.6	Required	The Standard Library input/output routines shall not be used		Pass
R.21.7	Required	The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used		Pass
R.21.8	Required	The library functions abort, exit, getenv and system of <stdlib.h> shall not		Pass
R.21.9	Required	The library functions bsearch and qsort of <stdlib.h> shall not be used		Pass
R.21.10	Required	The Standard Library time and date routines shall not be used		Pass
R.21.11	Required	The standard header file <tgmath.h> shall not be used		Pass
R.21.12	Advisory	The exception handling features of <fenv.h> should not be used		Pass
Resources				
R.22.1	Required	All resources obtained dynamically by means of Standard Library functions shall be explicitly released		Pass
R.22.2	Mandatory	A block of memory shall only be freed if it was allocated by means of a Standard Library function		Pass
R.22.3	Required	The same file shall not be open for read and write access at the same time on different streams		Pass

R.22.4	Mandatory	There shall be no attempt to write to a stream which has been opened as read-only		Pass
R.22.5	Mandatory	A pointer to a FILE object shall not be dereferenced		Pass
R.22.6	Mandatory	The value of a pointer to a FILE shall not be used after the associated stream has been closed		Pass