

References: [1] Advanced Architecture for Protocol Engineering Guidelines Revision 1.0

Code compiled using compiler	<compiler name>	<Version>
Compilation flags used		
Does code compile, without any error and warnings. If there are warnings, please mention those warnings. It is preferred that code is compiled with Insure also.(Optional)	Code compiles without any errors	
Readiness Index	-100.00%	

Guidelines Checkpoints				
Naming Conventions Checkpoints				
S. No.	Checkpoints	Yes/No/NA (Originator)	Comments (Originator)	Yes/No/NA (Reviewer)
1	<p>Are all files named according to Guidelines followed? According to AAPE, All files should have a name in the format &lt;name&gt;&lt;sep&gt;&lt;ext&gt;, where, &lt;name&gt;: alphanumeric characters up to eight characters in length, &lt;sep&gt;: is a period, &lt;ext&gt;: is an alphanumeric extension of up to three characters in length.</p> <p>All characters shall be in lowercase.</p> <p>The &lt;name&gt; field of the file should be split to represent the stack entity of the protocol suite and the module. If any characters are left and if multiple files are within a directory, the remaining length of the &lt;name&gt; can be used to identify the files. For example, a file of the Control protocol suite (of V5) belonging to the FSM module of ISDN-BA shall be named as: ctfsmib.c. ct: Control Protocol, fsm: FSM Module, i: ISDN, b: BA.</p> <p>Files that are common across protocols in a protocol suite shall bear the protocol suite prefix in their names. For example, v5global.h is a file common to all V5 protocols containing globally declared and accessible symbols and definitions.</p>	Yes	File names are given as per guidelines	Yes

2	<p>Are all symbol names follow the guidelines? According to AAPE, All symbols should be 32 characters or lesser in length.</p> <p>Underscore should be used to improve readability.</p> <p>Do not use case intermixing.</p> <p>Abbreviations should be consistent across the code.</p>	Yes	Symbol names are follow the guidelines	Yes
3	<p>Are all functions named according to guidelines followed? According to AAPE, All function names shall have a prefix that gives the stack entity name code and the module name code. It shall be of the following form:</p> <p>&lt;x&gt;_&lt;y&gt;_&lt;z&gt;</p> <p>Where, x: protocol stack entity code (not more than 4 chars)</p> <p>y: module name code (not more than 3 chars)</p> <p>z: rest of the function name</p>	Yes	Functions names are given as per guidelines	Yes
4	<p>Are all variables named according to guidelines followed? According to AAPE, All global variables should be prefixed with a unique prefix. This prefix represents the stack entity.</p> <p>All static variables should be prefixed with a unique prefix. This prefix represents the stack entity and module.</p> <p>All pointer variables should be prefixed with p_.</p> <p>A global (static) variable that is a pointer shall be prefixed with p_&lt;x&gt;_, where, &lt;x&gt; is the stack (module) prefix.</p>	Yes	Variable names follow the guidelines	Yes
5	Are all constants, literals, enumerations, MACRO names are in Uppercase?	Yes	All MACROS are in uppercase	Yes
Coding Style Checkpoints				
File Organizations				
S. No.	Checkpoints	Yes/No/NA (Originator)	Comments (Originator)	Yes/No/NA (Reviwer)

1	Does each header file contain provisions to counter erroneous multiple inclusions? For example, included files may include other files. To avoid multiple inclusions, protections are added to detect an already loaded include file. #ifndef SAMPLE #define SAMPLE..... #endif	NA	Header files are included only once.	NA
2	Do all files have proper and complete file headers according to guidelines followed?	Yes	All files use complete file headers	Yes
<b>Code Commenting &amp; Indentation</b>				
3	Is it ensured that any commented out source code is removed and no #if n source code (n is any number) is present?	Yes	Commented source code is removed	Yes
4	Is there any hack /offensive code? If yes, is it commented appropriately with proper justification or not?	NA	no offesnsive code is present	NA
5	Do the comments correspond completely to the code and comments are present for all complex legs of the code. Comments in block format should precede the statement/ group of statements. Block – end comment is given for every block where there are more than 10 statements.( Refer Coding guideline for change)	Yes	Comments are placed	Yes
6	Are all global variables declared in a module bear the comment explaining the intent, functionality and modified by module description?	NA	Global variables are not used	Yes
7	Is the code indentation consistent across all files? Also, Comment style used should be same. All horizontal lines of code should be within limit of 80 characters. Each line of source code has only one statement. If an expression is split in the code to the next line, it is always split before a unary operator and after a binary operator. In case of modification of an already existing source file, same convention need to follow. If changes are required, it should be applicable for the source file as a whole.	Yes	Indentation,comment style and all other conventions are followed	Yes
<b>Functions</b>				
8	Do all functions have complete and up to date Function Headers? According to AAPE, every global C function should have a header with function description. Groups of short, nearly identical static functions may be gathered under a single header. For more details refer appendix C.2 of AAPE.	Yes	upto date function headers are present	Yes
9	Is the size of more than 80% (preferably all) functions in the file reviewed less than 200 lines. If no, the code needs to be made more modular.	No	modular code is available	No
<b>Code Features</b>				
10	Are all global variable explicitly initialized? Preferably global variables should be minimized to make code MT-safe.	Yes	Global variables are not used	Yes

11	Is enough preference given to static variables in different files? Similarly functions used within the same files should explicitly be defined as static.	NA	Static variables are not used	Yes
12	Is the code free from magic numbers (hardcoding)? Only, "0 and 1" are allowed to be used directly in the code. Common usage of return values like SUCCESS and FAILURE should be 0 and 1 respectively.	NA	Code is free from magic numbers	NA
13	Is it ensured that all variables are initialized explicitly before using the value carried by them?	Yes	Variable initialisation is explicit	Yes
14	Is there any fall through statement in switch ? If yes is it commented?	NA	Switch is not used	NA
15	Do all if-else statement follow fully-bracketed syntax?	Yes	conditional statements are fully bracketed	Yes
16	Is it ensured that there are no floating numerical constants? Or if exist then comment is provided with justification.	Yes	No floating numerical constraints are there	Yes
17	Pointers (or references) have been used as function parameters wherever structures or arrays are required to be passed to the function.	Yes	Pointer are used when passing vectors to functions	Yes
18	Is code free from local variable definitions of large sized array and structures?	Yes	Code is free from large sized array local variables	Yes
19	Is NULL used for pointers, 0 for integers and '\0' for strings?	Yes	NULL is used accordingly	Yes
20	MACROs are defined with parenthesis enclosing the replacement text. MACROs definition doesn't end with a semicolon.If multiple replacement statements, the definition is enclosed by curly-braces.	Yes	MACROS are used as per specified	Yes
21	Does code have deterministic memory needs? Has developer estimated and mentioned the memory requirements in his code?	NA	Memory requirement is not identified	NA
22	Every system and library call (like calls to acquire memory, IPC etc) is checked for return values.	Yes	return values are checked	Yes
23	Basic data types of C shall not be used directly within the code. Instead a layer of types should be introduced based on the sign and length of the basic data type required.	NA	Not Applicable	NA

#### Other Checkpoints

#### Programming Checkpoints

S. No.	Checkpoints	Yes/No/NA (Originator)	Comments (Originator)	Yes/No/NA (Reviewer)
1	Are the conditional loops working correctly for the boundary conditions? (both underflow/overflow shall be checked)	Yes	loops are working correctly	Yes
2	Is code consistent with API document also? (If applicable)	NA	not applicable	NA
3	Are all possible messages / events handled in all states? (specially the error conditions/messages)	Yes	Error handling is done	Yes

4	Are all data structures chosen for optimum speed/memory behavior?	Yes	data structures choosen optimally	Yes
Quality Checkpoints				
S. No.	Checkpoints	Yes/No/NA (Originator)	Comments (Originator)	Yes/No/NA (Reviwer)
1	Is the tracability matrix updated for this piece of code? If yes, provide the path also.	NA	not applicable	NA
2	Has the updated RTM been sent along with source code for review? (RTM should also be a part of the deliverable for review)	Yes	RTM is available with source code	Yes
3	Is any tool used to do the static checking? (For example splint) If yes, provide the result also.	Yes	Results are available	Yes
4	Is insure/purify used?	No	Valgrind is used instead	No





















































Comments (Reviewer)
All the guidelines followed

All the guidelines followed	
All the guidelines followed	
All the guidelines followed	
All the guidelines followed	
Comments (Reviewer)	

Header files are managed properly
Files use proper file headers
No commented source code
No offensive code
Comments are provided
No global variables present
Code is properly indented
Function headers are up to date
lines of code more than 200
No glabal variables are present





Speed and Memory optimity is maintained
<b>Comments (Reviewer)</b>
Not applicable
Updated RTM is present along with source code
Results are present
Valgrind is used

















































































































