C++ Coding Standards and Guidelines Peer Review Checklist

Last Updated: 25 April 2016

Reviewer's Name:	Mark Shinozaki	**Peer Review Date:**	10/18/23
Project Name:		**Project ID:** Enter if applicable	
Developer's Name:		**Project Lead:**	
Review Files & Source code			
Code Approved			

The following check list is to be used in the assessment of C++ source code during a peer review. Items which represent the code being reviewed should be checked.

1. General Programming Standards and Guidelines

Refer to the OHD General Programming Standards and Guidelines Peer Review Checklist to assess the adherence to the OHD General Programming Standards and Guidelines.

2. C++ Programming Standards

2.1 Readability and Maintainability

yes Consistent indentation (3 or 4 spaces)

yes Consistent use of braces

No No tabs used

2.2 File Names

yes Header files and namespace files use suffixes: `.h`, `.H`, `.hh`, `.hpp`, or `.hxx`

yes Source files use suffixes: `.C`, `.cc`, `.cpp`, or `.cxx`

No `UpperMixedCase` is used for class or namespace file names

No `lowerMixedCase` is used for function file names

2.3 File Organization

No ____ Each file contains only one class declaration or definition except functors and static classes

No ____ File includes a brief description of the file after the *documentation block*

No ____ The content of the file is in the following order:

1. The preprocessor directives to prevent multiple inclusions in header files.
2. The *Documentation block* described in the *"OHD General Software Development Standards and Guidelines"*
3. A brief description of the file
4. Include files
5. `#defines` and Macros
6. The '`use`' directives in the source files but not in header files
7. Class or function declaration or definition

2.4 Include Files

yes ____ C++ standard library headers that have no extension are used

yes ____ New prefix `c` is used instead of the old extension `.h` for C standard header files

yes ____ The < > pair for library and system headers is used

yes ____ The " " pair for non-system (user defined) headers is used

No ____ No absolute or relative paths to point to the header files are used

yes ____ The system header files first in alphabetical order followed by the non system include files (including COTS includes) also in alphabetical order

2.5 Comments

No ____ The JavaDoc convention format is used for the documentation comment

yes ____ The C++ comment "//" style or the C style (/* ... */) is used for inline comments

2.6 Naming Schemes

yes ____ `namespace`, `class`, `struct`, template argument, and parameter names use uppercase letters as word separators with the first character capitalized

yes ____ Macro and `#defined` constant, `enum`, `union`, class static data member, and global variable names are all capitalized with underscore as separators

yes ____ Class methods and variable names use uppercase letters as word separators with the first character is not capitalized

yes ____ Private class data member names are prepended with the underscore, the rest is

the same as method names

No ___ `static const` data members are all uppercase

yes ___ `typedef` names reflect the style appropriate to the underlying type

yes ___ Class, struct, variable, and method names that differ by case only are not used

yes ___ C function names follow the *OHD C Programming Standards and Guidelines*

2.7 Class Design

yes ___ Class members are declared in this order: public members, protected members, private members

yes ___ Data members are properly protected (declared as private or protected)

yes ___ Classes (except functors and static classes) implement a default constructor, a virtual destructor, a copy constructor, and an overloaded assignment operator

yes ___ Static classes declare a private default constructor to prevent instantiation

2.8 Safety and Performance

yes ___ Type conversions have been done explicitly. The C++ set of casting operators `static_cast`, `reinterpret_cast`, `const_cast` and `dynamic_cast` have been used instead of C-style casting

yes ___ Global variables are not used except in rare cases and when used include an inline comment describing the reason for use.

yes ___ Dynamically allocated memory is deallocated when no longer needed

yes ___ There is no dangling pointers. Pointers are always tested for `NULL` values before trying to dereference them

yes ___ There is no hardcoded numerical values, const or enum type values are used instead

yes ___ Large objects are created on the heap

yes ___ The arguments specified in a function prototype are associated with variable names

3. C++ Programming Guidelines

3.1 Readability and Maintainability

yes ___ A space is put between the parenthesis and the keywords or the function names

yes ___ A space is put between variables, keywords and operators

yes ___ Pointers are named in some fashion that distinguishes them from other "ordinary" variables

yes ___ Parentheses are used in macros to ensure correct evaluation of the macro

no The `goto` statement is used very sparingly

3.2 User Defined Types

yes `static const` members are used instead of `#defined` constants

yes Proper `typedefs` are used instead of using templates directly

no `enum` is used to define a collection of integral constants

3.3 Variables

yes `const` correctness has been practiced

no All variables are correctly initialized

yes Local variables are declared near their first use.

yes The copy constructor is used to construct an object instead of the assignment operator (=)

3.4 Performance

yes `inline` functions are used instead of Macros

no The prefix form (`++i` or `--i`) is used instead of postfix form (`i++` or `i--`)

yes Pointer arithmetic has been avoided

yes Repetitive computations are reduced by only doing them once and saving the result in a temporary variable for future access

3.5 Class Design

yes Parts-of relation inheritance has been avoided