C++ Coding Standards and Guidelines Peer Review Checklist

Last Updated: 25 April 2016

Reviewer's Name:				**Peer Review Date:**	
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

_____Consistent indentation (3 or 4 spaces)

_____Consistent use of braces

_____No tabs used

2.2 File Names

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

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

_____`UpperMixedCase` is used for class or namespace file names

_____`lowerMixedCase` is used for function file names

2.3 File Organization

_____Each file contains only one class declaration or definition except functors and static classes

_____File includes a brief description of the file after the *documentation block*

_____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

_____C++ standard library headers that have no extension are used

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

_____The < > pair for library and system headers is used

_____The " " pair for non-system (user defined) headers is used

_____No absolute or relative paths to point to the header files are used

_____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

_____The JavaDoc convention format is used for the documentation comment

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

2.6 Naming Schemes

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

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

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

_____Private class data member names are prepended with the underscore, the rest is

the same as method names

_____`static const` data members are all uppercase

_____`typedef` names reflect the style appropriate to the underlying type

_____Class, struct, variable, and method names that differ by case only are not used

_____C function names follow the *OHD C Programming Standards and Guidelines*

2.7 Class Design

_____Class members are declared in this order: public members, protected members, private members

_____Data members are properly protected (declared as private or protected)

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

_____Static classes declare a private default constructor to prevent instantiation

2.8 Safety and Performance

_____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

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

_____Dynamically allocated memory is deallocated when no longer needed

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

_____There is no hardcoded numerical values, const or enum type values are used instead

_____Large objects are created on the heap

_____The arguments specified in a function prototype are associated with variable names

3. C++ Programming Guidelines

3.1 Readability and Maintainability

_____A space is put between the parenthesis and the keywords or the function names

_____A space is put between variables, keywords and operators

_____Pointers are named in some fashion that distinguishes them from other "ordinary" variables

_____Parentheses are used in macros to ensure correct evaluation of the macro

_____The `goto` statement is used very sparingly

3.2 User Defined Types

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

_____Proper `typedefs` are used instead of using templates directly

_____`enum` is used to define a collection of integral constants

3.3 Variables

_____`const` correctness has been practiced

_____All variables are correctly initialized

_____Local variables are declared near their first use.

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

3.4 Performance

_____`inline` functions are used instead of Macros

_____The prefix form (`++i` or `--i`) is used instead of postfix form (`i++` or `i--`)

_____Pointer arithmetic has been avoided

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

3.5 Class Design

_____Parts-of relation inheritance has been avoided