1. General code standard for CustusX

1. When implementing a workaround for a known issue, always add a comment to such effect in the code.
2. Use the "TODO" (all caps) keyword in comments that indicate parts of the code where more work should be done but is not required immediately. Avoid other keywords to the same effect.
3. Use C++ as the main language. Other languages are used for backwards compatibility and special uses only.

2. C++ source code standard

1. Files:
    1. C++ source files should have a .cpp extension.
    2. C source files should have a .c extension.
    3. C/C++ header files should have a .h extension.
    4. Use include guards in all header files of the form #ifndef uppercasefilename_H.
2. Comments:
    1. All code references in a header file shall be documented using javadoc-style Doxygen comments. This means /** ... text ... */.
    2. Comments should explain how something works when this is difficult to understand, and why an approach was chosen when this is not obvious.
    3. Focus on documenting classes and public methods. This is what defines the class to others.
3. Code Style
    1. When using Eclipse, set Code Style to BSD/Allman, line length 120. Format code using Ctrl-Shift-F.
    2. Normally use block grouping (ie { }) for if, while, for and do statements. Brackets are always on their own line, the first one indented similar to the preceeding code.
    3. Add a space after syntax elements like if, while, do and so on. Generally, if in doubt, add space.
    4. Use tab indentation for new lines of code, where tabs represent 2 or 4 spaces, except for tabulating continued lines, where spaces should be used. Users are free to chose the tab length they want.
4. Warnings
    1. All projects shall compile and link without warnings.
    2. Compile the code with as many warning flags as possible to catch errors. Always use the -Wall and -Wformat-security flags.
5. Names
    1. All comments and symbol names shall be in English.
    2. Filenames have the style <moduleprefix><classname>.cpp/h, for example sscVector3D.h
    3. Class names are camel case with first uppercase letter.
    4. Member variables in a class are usually prefixed with "m". The rest of the name is in camel case.
    5. Use accessors named getCamelCase() for get, setCamelCase() for set..
    6. All functions shall be named in lowerCamelCase().
    7. Smart pointer typedefs append Ptr to the class name. e.g. typedef boost::shared_ptr

FooPtr;

8. All new processes and libraries shall have its name prefixed ssc. Example: sscUtilities.so

6. Error handling and logging.
    1. After an error occurs, the module should be in a consistent state (no resource leaks, accepts input etc).
    2. Normally, the top-level code does not catch exceptions.
    3. Wrap calls that might throw exceptions within a try..catch block, unless you want to crash the process.
7. Modules/namespaces
    1. The main module in ssc is ssc.
    2. All modules shall have a namespace given by a shorthand version of its name.
    3. All namespaces shall be in lower case letters.
    4. Everything shall exist inside a namespace.
    5. Document namespaces with a doxygen comment (/** @namespace name blah blah blah */).
    6. Never use 'using' in header files. Use with care in cpp files.
    7.
8. Structure
    1. Remove unused code from production code. Do not litter the code with commented out code sections.
    2. Do not use globals, use singleton classes instead.
    3. Use pointer ('*') rather than reference ('&') to pass a return value from a function.
    4. Do not use arrays in public methods, use boost:array val; or similar classes in Qt or VTK instead.
    5. Avoid the use of the operator delete, use Qt/VTK-specific allocation methods, or smart pointers (e.g. boost::shared_ptr).
    6. Use const where possible.
9. Include statements
    1. Do not add unnecessary headers in a header file, use forward declarations instead.
    2. Organize includes starting with the most general library down to the closest neighbours, like this:
        1. std
        2. boost
        3. Qt
        4. vtk/itk
        5. ssc
        6. platform-dependent headers
        7. other modules
        8. current module
    3. .cpp files always include their own header file first.
10. Libraries.
    1. Write platform-independent code. The code will be run on Linux, Windows and MacOSX.
    2. The following libraries can be used freely:
        1. std
        2. boost
        3. Qt
        4. vtk
        5. itk

6. ssc
7. eigen
8. OpenCV

11. Testing
    1. Create unit tests for all new code.
    2. Run valgrind or similar on your test code.
    3. All public functions should check the validity of their input.
12. Numerical data
    1. Double precision ('double') floating point shall be used unless performance is critical and tested to perform better with single precision.
    2. Distances/Spatial data are given in millimeters, unless explicitly specified.
    3. Angles are given in radians, unless explicitly specified.
    4. Time are given in milliseconds, unless explicitly specified.
    5. Ratios are given in natural ratios [0-1] instead of percent [0-100] or bytes [0-255], unless explicitly specified.