General Coding Style

{{toc}}

In Pre-Stack Pro we use a mandatory common coding style.
This is to make it easier to understand, reuse and maintain the code.

The Pre-Stack Pro coding style is originally based on the ITWM coding style (see ITWM-Coding-Rules C/C++ - English)

Coding Rules

Naming

1. CamelCase is used basically for everything
   1. Classes start uppercase, e.g class MyCoolClass
   2. variables and functions start lowercase, e.g. void doSomething(); double averageValue; int m_currentIndex;
2. As prefix for PreStack Pro "PSPro" should be used, not "pspro" or "Psp" or "PSP".
3. Values of enums are all-uppercase. Example: enum Color {COLOR_RED, COLOR_BLUE, COLOR_GREEN};
4. Values of enums should all have a common prefix to show to which enum type they belong (example see above). Also this helps keeping the global namespace clean.
5. All function parameters start with an underscore, no other variables start with an underscore.
6. Static member variables should have a "s_" prefix, similar to "m_" for normal member variables and "g_" for global variables. This makes it obvious when dealing with static member variables.
7. Global and member variables and function arguments must have additionally to the "m_"/"s_"/"g_" prefix a prefix indicating their type. The following prefixes are mandatory:
   - p - pointer to something

Files

1. In general there is only one class per file. Exceptions are for trivial classes and similar things.
2. Files should be named exactly like the class they contain, including upper/lowercasing.
3. Filename suffixes to use for C++ files are ".h" for the headers and ".cpp" for the corresponding implementation files (i.e. no .hpp, .cxx, .cc, .cxx, .C etc.)

Formatting

1. Maximum line length is 120, without exceptions.
2. Placing curly braces: ITWM Coding style, 3.1, variant 1 is used, i.e. the opening curly brace is placed on a new line, not yet indented.
3. For indentation 2 spaces are used, no tabs.
4. Curly braces are used always after if/else/for/while/do, also for single statements. This makes accidential errors due to missing curly braces harder.
5. Only one statement per line, e.g.
   1. only one variable per line
   2. only one initializer in constructor-lists per line
   3. the block for if/for/etc. always on the next line
   4. exception: the ?-operator

General

1. "using namespace" must not be used in headers.
2. Using continue and break is ok.
3. Using goto must be avoided. If there are somewhere gotos, the labels, i.e. the targets for goto, must not be indented, i.e. they must start in column 1 of the line, and nothing else must be on the same line.
4. NULL is used for invalid pointers (and not 0).
5. When overriding a virtual function, put the keyword "virtual" again in front of the function declaration. This is not required from the language, but when looking at the code it makes it obvious that this is a virtual function.
6. All member function which don't modify the state of the object should be const.
7. Input pointer/reference parameters to functions should be const.
8. All QObject variables must have a name set using setObjectName(). The name should be the name of the variable (not of the class). It is recommended to use the macro SET_OBJECT_NAME(variable); from UtilsQt.h for that, it does that automatically.

9. Variables should be as local as possible

Comments

1. Doxygen comments are used for "API" documentation
2. "API" doucmentation, i.e. documentation which describes what a function or class does and how to use it, goes into the header file.
3. Every class should have a comment explaining what this class is about.
4. Do not put information which duplicates information from the code into comments, like "This is a public method that...". This is obvious, in the best case it just duplicates what you see, otherwise it contains wrong information.
5. If you think some code needs comments
    1. Think whether you can change the code so it becomes self-explaining, e.g. by better naming, temporary variables, splitting into functions, etc.
    2. if not, document what and why things are done, don't just repeat what the code itself says

Stuff to read

- How to write unmaintainable code
- Designing Qt-style C++ APIs

Files

codingrules_en_r02.pdf	79.2 kB	08/19/2010	Götz, Tobias