Code Standards

April 24, 2019

1 Code style

For code style, we use the standard QT style from QtCreator (see Code Style settings in Project Settings in QTCreator). Within namespaces, no additional indentation is used.

2 Code standards

2.1 Naming conventions

- Filenames, variable names, class names, etc are *camelCase*. Each word except the first is capitalized. Except for classes and structs, where also the first character is capitalized.
- Abbreviations in variable names should be avoided. Exception, common abbreviations like *json*.
- Header files have extension `.h`, source files have extension `.cpp`.
- Member variables are preceded with an underscore: `_variableName`.
- Avoid protected member variables, use private members and create getter and setter.
- Getter name is member variable name prefixed by `get`: `getVariableName()`.
- Setter name is member variable name prefixed by `set`: `setVariableName()`.

2.2 Headers

- Header safeguarding is achieved using `#pragma once`.
- Includes of system headers with `#include <file.h>`, includes of non-system headers with `#include "file.h"`.
- Headers are order from specific to general. First our own includes, then includes from Eigen, Google Test or other libraries and concluded by the most generic includes from the standard library.

2.3 Classes

- Each class has a header file (`.h`) and a definition file (`.cpp`).
- In definition files variables have names. These names match the names in the definition file.
- Constructors use the initialization list, and the constructor of parent classes as much as possible.
- Implementations of virtual functions in derived classes get the `override` identifier. `virtual` is therefore not repeated.

- The `final` specifier is used on classes `derived final:base`, if it does not make sense to derive classes from given class.
- Move and copy constructors are only explicitly added if they do not have default behavior.
- Be `const` correct; getters are const methods, arguments in setters are const, etc.

2.4 Arguments

- Pass scalar types by value in arguments.
- Const argument only when possible, so make sure to never copy a variable that was send by value in the arguments.
- Use constant references when passing object argument, as `const type&`

2.5 Miscellaneous

- All classes are defined within the appropriate library namespace.
- Namespace name is added as a comment to the closing brace.
- The only thing to be thrown is exceptions derived from `std::exception`.
- Do not throw exceptions from constructors or destructors.
- For initializing variables we use the braced initializer list.
- Use `nullptr` for null pointers, do not just write `0`.

2.5.1 Example of a class definition

```
// Basic description of class
#pragma once

namespace basin
{

class Derived : public Base
{
  public:
    Derived();
    void virtualMethod() override;

    type getVariable() const;
    void setVariable(const type& variable);

  private:
    type _variable;
};

}  // basin
```