Coding Conventions for Sierpi

Martin Schreiber

July 5, 2012

1 Purpose

Different developers working together have usually different coding conventions. Therefore an agreement to coding conventions has to be achieved when working together on one project for several reasons. One of the main reasons is to keep the code clean and thus to maintain a better code structure.

2 Coding conventions

2.1 Identation

Blocks and scopes have to be indented by using a single TAB for each identation.

2.2 Naming conventions

2.2.1 Types

Types are distinguished whether they are used as a

- class,
- class with templates
- typedef followed by a specialized class
- classes without templates
- atomic variables (int, char, float, ...)

When classes are defined, they are always prefixed with a capital letter 'C' denoting that they are a class. Also classes with template parameters are prefixed with the 'C'.

As soon as a class (with or without template parameters) is given as a template parameter or redefined via a typedef, the 'C' is replaced by a 'T' to account for a fixed type without necessity of template parameters.

2.2.2 Variable naming

- For atomic types (int, char, float, etc.), all variables have to be written using underscores and small letters.
- For class types, the variables have to be written without underscores with the following exceptions: For better understanding, it is allowed to extend those types with underscores and further text. E. g. cEdgeCommData_rightData_only. Variables which are derived from a class type starting with a capital letter 'C' (this should be usually the case), are expected to start with a small letter 'c'. E. g. instantiating a variable with the type 'CEdgeComm' would give a variable the name cEdgeComm.

2.2.3 Parameters for methods

parameters of methods are prefixed by i_, o_ or io_.

- i_ means that this parameter is accessed read/only (const).
- o_ is used to declare this parameter as being an output reference to write some output values.
- io_ is used to declare an input/output pointer or reference which is read and written.

Output parameters always have to be of type pointer. No references should ever be used for output parameters! Return values handed back to the calling method are still allowed and have so far no convention.

```
/**
 * some comment
 *
 * \return description of return value
 */
char foo(
    const int i_bar_var,
                ///< this is a totally useless variable
    const CSomeClass &i_cSomeClass,
                ///< input via referenced parameter
    CSomeClass *o_cSomeOutputClass
                ///< output via pointer parameter
) {
    while (true) {
        ...
    }
    return 42;
}
```

2.3 Template parameters

ALL template parameters have to be prefixed with a "t_" to differ between template types and other types.

2.4 Comments

Comments are one of the most important thing in writing code. Therefore as much comments as are necessary or being requested by other developers have to be written.

```
/**
 * comments preceeding functions should
 * follow the doxygen (www.doxygen.org)
 * code-style
 *
 * \return description of return value
 */
void foo(
    const int i_bar_var, ///< this is totally useless
    void *io_foo_var     ///< this variable is not
                         ///< able to drink beer
) {
    while (true) {
        ...
    }
}

constructor(
    int i_val,  ///< index
) :
  some_variable(i_val)
{
        ...
}
```