LUMA Coding Standards

Last update: 15th February 2017

Purpose

This document sets out the rules associated with formatting, design and organisation of code written in LUMA to ensure standardisation across the development team. In other words, new features written into the code should be developer-agnostic from an appearance perspective. These standards should make use of best practice wherever possible to ensure first and foremost that code is readable, especially by those developers with less C++ experience.

Standards

Macros

All LUMA pre-processor macro definitions should be capitalised, be prefixed by L_ and have words separated by and underscore. Definitions should not have numbers in their names. E.g.

```
L_NO_FLOW
```

Line Length

Wherever possible, lines should not exceed 81 columns to maximise readability.

Comments

Single line comments should leave a space between the start of the comment and the double slash. Example

```
// My comment
```

Multi-line comments should use the “java-style” syntax with repeated asterisks. Example:

```
/* My multi-line
 * Comment */
```

Brackets and Braces

Braces, used to define code blocks should always be aligned and should be started on a new line for clarity. This principle should also extend to for loops and other commonly used constructs. Example:

```
void myMethod(…)
{
```

```
// etc.
}
```

No extra white space is needed when using parentheses and should be used in-line unless the expression they encompass is large or exceeds the line length guidance. In such cases, the parentheses may be split across lines like braces for clarity. Example:

```
void myMethod(arg1, arg2,…
    …argn…

    …argnn);
```

If possible, statements should be aligned in this way

```
if
(
    cond1 &&
    cond2 &&…
   …condn
)
{
// etc.
}
```

White Space

White space is encouraged to improve readability. In particular, leave a space after the comma in multi-argument method calls and also in between numerical operators. Example:

```
N_lim * M_lim;

myMethod(a, b, c);
```

Control Flow Compact Form

The compact form of single line body control flow statements (without the braces) can be used if it improves readability and obeys the line length guidance but should be on separate lines to facilitate debugging. Example

```
if (short condition)
   doSomething();
else
    doSomethingElse();
```

Ternary syntax may also be used e.g.

```
Variable = (condition) ? value1 : value2;
```

Initialiser Lists

Initialiser lists should be used wherever possible to simplify constructors. For readability, the preceding colon should be placed on the next line. Example:

```
MyClass::MyClass()
    : member1(arg1)
{
```

```
// etc.
}
```