Coding Style

How to write code that not only you can read…

A proposition:

CamelCase

- Since space bars are not allowed, we have to concatenate words like „space ship“.
- Either „space_ship“ or „SpaceShip“
- The latter is so called CamelCase
- We mostly use CamelCase whenever possible

Files, General

- Almost every class has a header and a source file.

✂→Mapping: File name <- -> Class name

- Ex: SpaceShip.cc <- -> SpaceShip
- Try not to exceed 2000 lines

Files, Headers

- Always use an include guard:

```
#ifndef _SpaceShip_H__
#define _SpaceShip_H__
… // File content
#endif /* _SpaceShip_H__ */
```

- Try to use forward declarations in LibraryPrereqs.h:

  `class SpaceShip;`

  → also minimises header file dependency.

- Never write `using namespace foobar;` in a header.

Source Files

- First include file should always be the correponding header file.

- E.g. in SpaceShip.cc:

```
#include „SpaceShip.h“
#inlcude „more.h“
```

- Again: Try to minimise dependencies to reduce compile time.

Template definition

- When using templates or longer inline functions, declare them in the class and define them below.

```
template <class T>
class SpaceShip
{
    void myFunction();
};
template <class T>
void SpaceShip<T>::myFunction()
{  }
```

Naming Classes

- All class names should start with a capital letter
- CamelCase again

```
class SpaceShip { };
struct MyType { };
```

Naming Variables

- CamelCase, but starting with lower case.
- Ending with „_“ for class members and „_s“ for static class members.
- Try to use „this->“ when refering to a class variable.
- `int a = 8; char b = ‘ c‘;`

  Meaning of a and b?

static const

- Global variables that are const have no real linkage in C++ meaning that they only exist at compile time.
- `const int CONSTANT_NUMBER = 8;`
- Use all upper case names to declare constant variables.
- In a class, use `static const int myInt = 9;`
- Never use macro defines for global constants if you can help it. C++ doesn‘t need that anymore.

Naming Functions

- CamelCase, starting with lower case letter
- Try using a verb:

  `calculateTheLowerBoundary()`

Enumerations

- Enums leak their member into its corresponding namespace.
- Please use the following syntax instead:

```
namespace Colour // Upper case
{
    enum Enum // or in combination with ns
    {
        Red,
        Blue
    };
}
```

Indentation and white spaces

- 4 spaces per tabulator, never use tabulators
- White spaces in operators:

```
if (a + b < c)
```

Magic Numbers

```
if (threshold > 135.0)
    doSomething();
```

- What is 135.0 anyway?
- Try to name such numbers with a constant variable. This makes code more readable and configurable.

Data encapsulation

- Make all member variables private
- Use set and get functions to change and access them.
- This optionally gives you more control
- If the code is inline in the class file, there is really no preformance loss at all

Control Statement Layout

```
if (condition)
    doStuff();
if (condition2)
{
    doStuff();
    doMoreStuff();
}
else
    doSomeOtherStuff();

for (int i = 0; i < size; ++i)
{
}
```

The trap

```
if (condition1)
    if (condition2)
        doSomething();
else
    doSomethingElse();
```

- Always use curly brackets when having nested control statements.

Class Declarations

```
class MyClass : public BaseClass
{
    public:
    protected:
    private:
}; // don‘t forget that semicolon!
```

Code Review

- Handing over your code to someone else can eliminate errors.
- This also enforces you to write documentation as you code.

http://www.orxonox.net/wiki/development

Documentation

- In-code comments: Important for understanding control flow
- Explain your ideas if they are not obvious
- Write additional documentation for functions, variables, classes and even files.
- We use Doxygen to ease up the task. Fully comparable to javadoc.

Documenting a Function

```
/**
@brief
    A brief description.
    More brief description.

    Details follow here.
    Maybe an example or something.
@param param1
    Parameter description for parameter named 'param1'
@param param2
    Parameter description for parameter named 'param2'
@return
    Describe what the function returns
@note
    Something important to remember.
@author
    The mighty whoever
*/
returntype functionname(type1 param1, type2 param2)
{ ...
```

More documentation

- Also document member variables. This can easiest be done like this:

```
int aNumber_; //!< This is a number
```

- Classes can be documented above the declaration.
- The generated HTML documentation can be found under ‚Doxygen‘ in the Development section of our wiki.
- More information:

  http://ww.orxonox.net/wiki/Doxygen