Coding Style

General Recommendations

The main goal of these recommendations is to improve the readability, understanding and maintainability of the source code. Violation of these recommendations is allowed only if it enhances the readability of the source code.

Header Files

Every .cpp file should have an associated .h file. Files should be saved in the same directory. An exception to this is the main.cpp file, which does not have to have an associated .h file. Other small files containing unit testing can also be excluded from having an associated .h file.

All .h files must be included in the ResourcePath.hpp file. This provides a record of all header files in one place. Every .cpp file will only need to contain:

```
	#include “ResourcePath.hpp”
```

Include Statements

Include statements should be sorted and grouped by their hierarchical position in the system, with low level files included first. An empty line should be placed between groups of include statements. Include statements should be located at the top of a file only.

Example:

```
	#include<iostream>
	#include<fstream>

	#include<cmath>
	#include<algorithm>
```

The #define Guard

All header files should have the #define guard to avoid duplicate declarations. Formatting for the #define guard should be

```
<PROJECT>_<PATH>_<FILE>_H_
```

Naming Conventions

1. Names representing types must be in mixed case starting with upper case.
2. Example: `Name, FileName`
3. Variable names must be in mixed case starting with lower case.

   Example: `name, filename`
4. Constants must be all upper case using underscore to separate words.

   Example: `MAX_NAME, PI`
5. Names representing methods should be verbs and written in mixed case starting with lower case.

   Example: `getName(), computeArea(), setName()`
6. Names of classes should begin with an uppercase letter.

   Example: `Class MyClass`

Variables

1. Variables must be initialized upon declaration.
2. Variable names must be clear and meaningful.
3. Class variables must always be declared as private members.

Functions

Functions should be kept as small as possible. If you have a feeling a function is too long, then it probably is; consider separating it into smaller parts.

There is no hard limit to the size of a function, as at times it is unavoidable that a function grow in size- this can be acceptable if there is no alternative method of reducing it into smaller parts.

Tabs, Spacing and Indentation

Tabs must be set to four characters.
The bracing must follow the style whereby the opening brace begins on the same line of the statement and ends on a new line.

Example:

```
If (a < b) {
     a = b;
}
```

Classes must follow the same format for bracing, with the opening brace on the same line of the class name and the closing brace on a new line. The labels for *private:* and *public:* are not indented.

Example:

```
Class MyClass {
public:
     MyClass();  // Constructor
     ~MyClass(); // Destructor
     void myMethod();
private:
     int year;
};
```

One exception to this bracing style is permitted- inline functions may have their closing brace on the same line as their opening brace.

Code Comments

Comments should be used to explain what a section of code does if it is not clear. It is preferable that if the code is too complex to understand that it be re-written.

Single line "//" commenting is to be used, even for multi-line comments. Comments must include a space immediately after "//", and should start with an upper case character and end with a period, as in standard English.

Block commenting "/* */" should only be used for commenting out sections of code for debugging purposes.