Chapter 1

Coding Standards

It is important that coding of an application is carried out in a consistent way. Consistency has several advantages:

- It can help to highlight logical errors in the code[1.1];
- It makes it easier for new developers or casual 'dabblers' to get familiar with the code;
- It supports the principle of 'discovery by element of least surprise'[1.2];
- The code is presented more neatly making it easier to read and understand;
- The code is self documenting.

There are numerous useful resources relating to coding standards [****** for example that book I bought in Oxford *****]. The best advice most people give is that its not so important how you define your coding convention, more that you have and follow one in the first place. In order to achieve that (a coding standard that people actually follow), a few basic criteria were defined:

- The rules should be simple and easy to remember;
- The rules should not place an onerous burden on the developer;
- The rules should promote consistency and clarity;
- The rules should leave room for programmer creativity.

In the sections that follow, I describe the conventions followed for the OMGUI project.

[1.1] For example by being able to distinguish pointers easily by their prefix.

[1.2] For example if the developer is able to guess what method names are because a consistent system is used, he can save time by not needing to know every API off by heart.

1.1 Naming Conventions

1.1.1 Classes

Names

Classes in OMG[1.3] begin with Omg and are formed using mixed case.

```
Examples:
        OmgPoint
        OmgDataSet
        OmgLocalClient
```

The rationale for using the Omg prefix is explained on the Trolltech Qt Naming Page[1.4].

Members

Class member names begin with a lower case "m" and are formed using mixed case.

```
        MapCanvas mMapCanvas;
        CurrentExtent mCurrentExtent;
```

In the case where a class member is a pointer, use the letter p to indicate as much:

```
        MapCanvas * mpCanvas;
        Model * mpModel;
```

All class members should be private. **Public class members are STRONGLY discouraged**

Function parameters

Prefix all function parameters with 'the' (or 'thep' in the case of pointers). I have only ever seen this suggested in one C++ book, but I **really** like this naming scheme:

```
bool Foo::Bar(Dog theDog, Cat * thepCat)
{

}
```

[1.3]The OMG prefix is a registered prefix with Trolltech (see http://www.trolltech.com/developer/notes/naming/?searchterm=omg). Having the Omg prefix registered means we can integrate code from other Qt based projects without worrying about naming conflicts.

[1.4] http://www.trolltech.com/documentation/naming.html

Local Variables

As above but use 'my' prefix:

```
bool Foo::Bar()
{
  Bunny myBunny;
  Horse * mypHorse
  ...
}
```

Accessor and Mutator Functions

Class member values should be obtained through **accesssor functions**. The function should be named **without** a *get* prefix. Accessor functions for the two private members above would be:

```
occurrence()
maxValue()
```

Mutator functions should follow the same convention but **must** be prefixed by the word *set*. For example:

```
setOccurrence()
setMaxValue()
```

Functions

Function names begin with a lowercase letter and are formed using mixed case. The function name should convey something about the purpose of the function (usually in the form of a verb).

```
updateMapExtent()
setUserOptions()
```

1.1.2 Qt Designer

Generated Classes

Classes that are generated from Qt Designer (ui) files should have a "Base" suffix. This identifies the class as a generated base class.

```
Examples:
OmgPluginMangerBase
OmgEnvironmentalDataPanelBase
```

Dialogs

All dialogs should implement the following:

* Tooltip help for all toolbar icons and other relevant widgets * WhatsThis help for ”’all”’ widgets on the dialog * An optional (though highly recommended) context sensitive ”Help” button that directs the user to the appropriate help page by launching their web browser

1.1.3 C++ Files

Names

There should be only one class (or struct) per file. A struct may well be upgraded to a class at some point in future so treat it like a class! C++ implementation and header files should be have a .cpp and .h extension respectively.

Filename should be all lowercase and, in the case of classes, match the class name. For example:

```
Class OmgDataset source files are:
omgdataset.cpp and omgdataset.h
```

Standard Header and License

Each source file should contain a header section patterned after the following example:

```
/**************************************************************************
    omgfield.cpp - Describes a field in a layer or table
     ---------------------------------------
    Date                 : 01-Jan-2013
    Copyright            : (C) 2013 by \emph{Your Name}
    Email                : \emph{your.email@address}
/**************************************************************************
 *                                                                        *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                        *
 **************************************************************************/
```

SVN Keyword

Each source file should contain the Id keyword. This will be expanded by SVN to contain useful information about the file, revision, last committer, and date/time of last check-in.

Place the keyword right after the standard header/license that is found at the top of each source file. After it has been expanded by cvs it will look like this:

```
/* $Id: coding_standards.tex,v 1.2 2006/08/02 19:50:08 username Exp $ */
```

1.1.4 Editing

Any text editor/IDE can be used to edit OMG code, providing the following requirements are met.

Tabs

Set your editor to emulate tabs with spaces. Tab spacing should be set to 2 spaces.

Indentation

Source code should be indented to improve readability. There is a .indent.pro file in the OMG (”’TODO add this file”’) src directory that contains the switches to be used when indenting code using the GNU indent program. If you don’t use GNU indent, you should emulate these settings. The correct settings can be achieved in vim by adding the following lines to your /.vimrc file:

```
set cino=>2
set softtabstop=4
set tabstop=2
set shiftwidth=2
set autoindent
set smartindent
set showmatch
```

Braces

Braces should start on the line following the expression:

```
if(foo == 1)
{
  // do stuff
  ...
}
else
{
  // do something else
  ...
}
```