Bob's Concise Introduction to Doxygen*

by Robert S Laramee†
Visual and Interactive Computing Group
School of Computer Science
University of Nottingham, UK
http://www.cs.nott.ac.uk/blaramee/

September 28, 2021

1 Comment Standard

The general rules for commenting your source code are as follows:

1. File, class, and method comments go into the header (`.h`) files (as opposed to the implementation `.cpp` files).
2. Header files begin with a comment containing
   (a) `\file` the name of the file, e.g., RSL_Triangle.h
   (b) `\author` the author of the file, e.g., Robert S. Laramee
   (c) `\date` the date this file was created, e.g., 29 Oct 2009, and optionally
   (d) `\see` for names of related files.
   (e) `\brief` for brief descriptions of classes. Add an empty line followed by a more detailed description of your class.
3. All multi-line comments start with the starting sequence `/**` on a line by itself and end with the terminating sequence `*/` on line by itself.
   Exception: single line comments.
4. All classes are accompanied by a meaningful description.of one or more sentences. The general responsibilities are given.
5. All methods have `@param`, and `@return` tags.
   Exception: there are no method parameters or nothing is returned.
6. Public methods appear first, then private methods, and finally private data members with explicit `public:` and `private:` identifiers at the beginning of each group.
7. Within each public and private group of methods, methods appear in alphabetical order.
   Exception: accessor methods (`Get()` and `Set()`) are at the top.

*Started on Friday 14 Sep 2001. Prepared with LaTeX
†robert.laramee "at" nottingham.ac.uk

2 Diagrams

Doxygen has the amazing ability to automatically generate class hierarchy and collaboration diagrams (also called a dependency graph) from your software. Doxygen uses a tool called *dot* from GraphViz to generate advanced diagrams and graphs. GraphViz is an open-source, cross-platform graph drawing program and can be found at: `http://www.graphviz.org/`.

After downloading and installing the dot tool set `HAVE_DOT` to `YES` in your doxygen configuration file to use it. For more details visit `http://www.doxygen.org/` and click on **Manual** and then click on **Graphs and diagrams**.

3 Outputting the Source Code

Doxygen provides the user with the option of outputting the source code along with the HTML output. This option is extremely useful for code reviews. Set the `SOURCE_BROWSER` and `INLINE_SOURCES` tags both to `YES` in your doxygen configuration. For more details visit `http://www.doxygen.org/` and click on **Manual** and then click on **Getting started**.

4 Doxygen Examples

These comment conventions are based on doxygen. Preparing source to be processed by doxygen is easy. Comment blocks delimited by `/**` and `*/` must be added before each item for which you wish to generate documentation. Summarizing, source code is documented with the following:

1. each class definition –explaining the purpose of the class
2. each member function –explaining what the function does
3. each member variable –explaining what the variable means
4. each type definition (enums) –explaining what the type represents

Within a comment block you can use tags to explain function parameters, return values, and more. The three most important doxygen tags are:

`\param` –explains function parameter
`\return` –describes return value
`\see` –a reference to a function name, variable name, document, or URL.

4.1 Here is an example of how a method is commented in doxygen:

```
/**
  * Calculate the center point of a triangle.  This method
  * has been ported from the Visualization Toolkit.
  * \see www.kitware.com, vtkTriangle.h
  *
  * \param coord0 the 1st x,y,z coordinate of a triangle
  * \param coord1 the 2nd x,y,z coordinate of a triangle
  * \param coord2 the 3rd x,y,z coordinate of a triangle
  * \return the triangle center as an x,y,z coordinate
```

```
  */
Coord3D< Float > GetTriangleCenter(
                        const Coord3D< Float >& coord0,
                        const Coord3D< Float >& coord1,
                        const Coord3D< Float >& coord2);
```

4.2 Here is an example of how a class is documented in doxygen:

```
/**
  * SU_Triangle is intended to provide methods to perform
  * common operations on a triangle in 3D space.  It is
  * intended to be highly legible, highly generalized, and
  * highly reusable.  Some methods have been ported from The
  * Visualization Toolkit.
  *
  * \see http://www.kitware.com
  **/
class SU_Triangle {
...
};
```

It is also possible to use HTML tags in the comment blocks to create elaborate lists, bold and italic text etc.[1] When documenting your code please keep in mind that other developers will have to understand what your classes and methods do by reading the documentation generated from your comments. They do not have any knowledge of your implementation, thus your comments must explain everything that helps in understanding your code. The examples show how to comment a class so that useful documentation can be generated.

[1] For detailed information please see `http://www.doxygen.org/`.

4.3 A Complete Header File Example:

```
/**
 * \file   hello.h
 * \author Sitsofe Wheeler
 * \date   29 Oct 2009
 * \see    ``The OpenGL Programming Guide'' Fifth Edition Page 18
 *
 * \brief Header file for simple "hello world" OpenGL rectangle example.
 *
 * A detailed description of the Hello class goes here.
 * This is a Class to create an OpenGL triangle.  It expects glutInit
 * to have been called.
 */
class Main {

   public:
   /**
    * Creates the GL window (with hello as the title) and starts
    * the event loops.
    * \see glutInit()
    *
    * \param argc -the number of parameters passed to the program.
    * \param argv -an array of cstrings containing parameters
    *             passed to the program. First parameter is the name the
    *             program was called with.
    */
   static void main(int argc, char **argv);

   private:
      /**
       * Initialise the OpenGL state machine.
       */
      static void init(void);

      /**
       * Renders the rectangle.
       */
      static void display(void);
};
```

4.4 A Complete Class File Example:

```
/**
 * @file   hello.cpp
 * @author Sitsofe Wheeler
 * @date   29 Oct 2009
 * @see    hello.h for class documenation.
 *
 * Simple "hello world" OpenGL rectangle example.
 */

#include <GL/gl.h>
#include <GL/freeglut.h>
#include "hello.h"

void Main::init(void) {

   /** select clearing (background) colour */
   glClearColor(0.0, 0.0, 0.0, 0.0);

   /** initialize viewing values */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

void Main::display(void) {

   /** clear all pixels */
   glClear(GL_COLOR_BUFFER_BIT);

   /**
   * Draw white polygon (rectangle) with corners at
   * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
   */
   glColor3f(1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f(0.25, 0.25, 0.0);
      glVertex3f(0.75, 0.25, 0.0);
      glVertex3f(0.75, 0.75, 0.0);
      glVertex3f(0.25, 0.75, 0.0);
   glEnd();

   /** Don't wait! Start processing buffered OpenGL routines */
   glFlush();
}

void Main::main(int argc, char **argv) {

  /**
  * Declare initial window size, position, and display mode (single
  * buffer and RGBA). Open window with "hello" in its title bar. Call
```

```
  * initialization routines.  Register callback function to display
  * graphics.  Enter main loop and process events.
  */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(250, 250);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("hello");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}

/**
 * Program entry point. Initialise glut and call Main class.
 *
 * @param argc -the number of parameters passed to the program.
 * @param argv -an array of cstrings containing parameters passed to the
 *          program. First parameter is the name the program was called
 *              with.
 * @return -0 on success.
 */
int main(int argc, char **argv) {

   Main::main(argc, argv);
   return 0;
}
```

5 Acknowledgements

Understanding undocumented code is difficult, time-consuming, and can even be impossible. Thanks to Sitsofe Wheeler for valuable discussions on this topic. Feedback on this document is not only welcome but encouraged. Please send correspondence to the first author. These code comment conventions are to be used in conjunction with the coding conventions [1].

References

[1] R.S. Laramee. Bob's Concise Coding Conventions (C^3). *Advances in Computer Science and Engineering (ACSE)*, 4(1):23–26, 2010. (available online).