MMITSS Comment Style Guide

# Purpose:

The purpose of this document is to describe the amount of documentation and markup used in the MMITSS codebase in order to create deployable and maintainable source code. This document can be changed with discussion by the MMITSS team. All source files should maintain the style described here.

# Markup:

This project uses a tool called Doxygen to read comments from code and create HTML documentation that can be distributed or used locally. The documentation and download for doxygen can be found [here](http://www.stack.nl/~dimitri/doxygen/manual/index.html). The documentation is a great reference for the markup that will be used on this project.

Doxygen supports many different styles of markup. This project uses JavaDoc. A simple example of JavaDoc is shown in figure 1.



Figure 1.

The basic structure of JavaDoc is to have a second asterisk (\*) after your comment opener (/\*). Any additional lines begin with an asterisk and the final line ends the comment block (\*/).

Traditionally, there is a brief description followed by a more detailed description in remaining lines. JavaDoc has a configuration setting that automatically makes anything prior to the first dot (.), that is followed by a space or newline, the brief description. This project utilizes this setting to make the comments read clearer in the code. The configuration setting is titled JAVADOC\_AUTOBRIEF and is automatically set to NO. In order to have Doxygen recognize a brief comment, the configuration setting must be set to YES. This is done in the “Expert” tab of the Doxygen GUI. If AUTOBRIEF is not used, the @brief setting must be applied to the front and a blank line must occur before the beginning of the detailed description to signal the beginning of a new paragraph. Figure 2 shows an example of what a comment using AUTOBRIEF looks like in the code and figure 3 shows the Doxygen HTML output associated with the comment in figure 2.

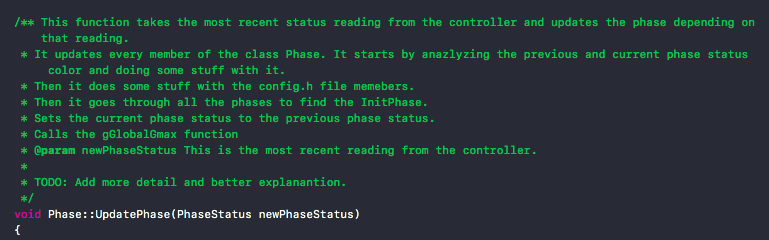
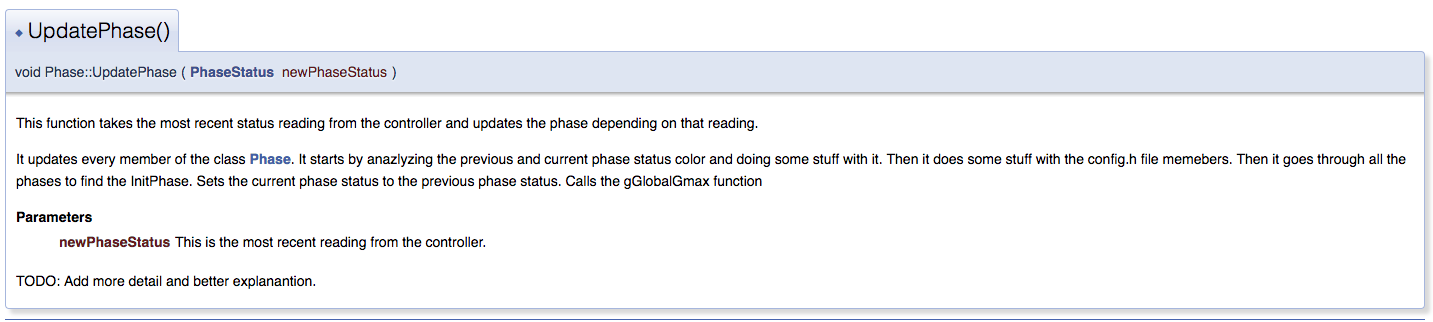


Figure 2.

Figure 3.

The Doxygen manual explains structural commands in detail with the different types you can use and when you must use them. The documentation primarily uses the \(command name) style. However, to maintain consistency with JavaDoc, this project uses the @(command name) style.

In order to escape a structural command, a new paragraph must be created by leaving a blank line after the structural command and related comment. After the blank line, anything else written will not be tied to the structural command. This is displayed in Figure 2 after the structural command of @param is used.

If a detailed description lends itself to multiple paragraphs, a new paragraph can be started by leaving a blank line between the end of the first paragraph and the beginning of the new paragraph.

# Comments:

Every file will begin with a heading comment at the top flagged with @file. This comment will give a description of what the file does, why it exists, and to what degree it interacts with other pieces of the system. This is not meant to be a detailed description of everything in the file, but rather a brief synopsis of the file’s purpose. The example given in figure 4 shows a very succinct description of a header file. Other files may have longer descriptions.

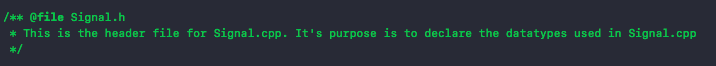


Figure 4.

Every class should begin with a comment explaining what the class is and why it exists. This comment must be flagged with @class. It also should include why each member exists and what it is. The member functions will generally not be included in this comment block. They will be described above their definitions and Doxygen will link them to the class in the HTML output. The member attributes of the class will be commented in the class comment block.

Every function should begin with a comment explaining what the function does, why it exists, and how it fits into the bigger picture. It should include what each parameter that is passed in means and what are acceptable values to be passed in. It should also describe what value (if any) is returned and what the acceptable values are. Any lines of code where the functionality of the line is not clear from the code alone should be followed by an in-line comment that describes the functionality. In-line comments should only be used to explain specific lines of code and, therefore, shall be omitted from the documentation. These comments will not be written in JavaDoc. Figure 5 shows an example of a function heading comment. Figure 6 shows the HTML output for comment in figure 5.

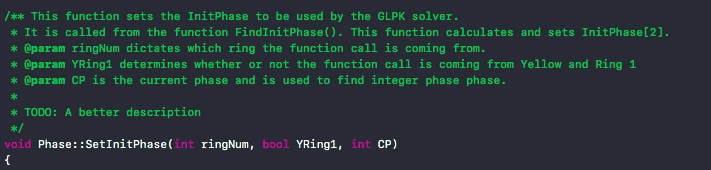


Figure 5.

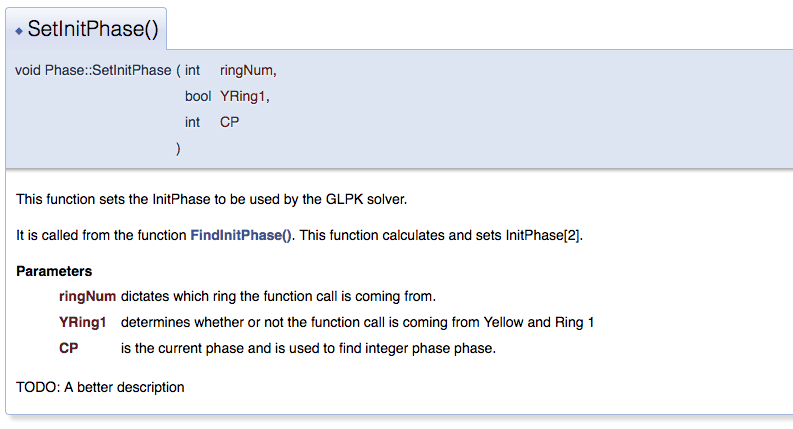


Figure 6.

Much of the MMITSS code is algorithmic. When a file contains an intricate algorithm, part of the detailed description of the main function should contain an overview of the algorithm and what it is. The details will be in the individual functions that actually implement the various pieces of the algorithm. This description is to give someone glancing at the documentation an idea of what is happening.