# Terms and Definitions

See Programming Terms and Definitions document.

# End User

When programming, keep in mind who your end users are.

* Who are the programs for? Not what are the programs for.
  + End User – Robot and people running the robot.
  + Other Programmers:
    - Co-programmers
    - Future programmers
    - Maintainers

# Comments

The best way to make your code ready for other programmers is to properly comment it. Some people say that code should be self-documenting. Only in the best or simplest case is this true. Even then, the code is often self-documenting to the original programmer.

## Doxygen

Doxygen is a tool that turns comments into documentation. Javadoc comments are basically a subset of Doxygen comments. Doxygen can generate documentation is different formats: HTML which looks like Javadoc output, LaTex, man pages, RTF (Rich Text Format), XML, and Docbook.

While Javadoc is limited to Java only, Doxygen can be used on most languages including, and not limited to, Java, C++, C, FORTRAN, PHP, and Python.

### Doxygen Comments

Doxygen reads doxygen comments for the source of the generated documentation. Doxygen uses various commenting styles that each of its supported languages use . Like Javadoc, the comment style:

/\*\*

\* … text …

\*/

The double asterisk denotes the beginning of a doxygen comment. This style also works for C++ and other languages that use C-Style comments. Other comment styles can be used to denote doxygen comments. See <https://www.doxygen.nl/manual/docblocks.html> for more details.

Doxygen looks for commands within the doxygen comment that tell how that section of the comment is to treated. Commands begin with the character ‘@’ or ‘\’. For best practices, chose one of the two characters and use only that one in your code. See <https://www.doxygen.nl/manual/commands.html> for list of all the Doxygen commands.

### Where to use Doxygen Comments

There are no hard rules for when and where to place doxygen comments though there are a few places where they are most useful.

Note, the @brief command can be omitted if javadoc\_autobrief, which Doxygen calls BRIEF\_MEMBER\_DESC, is enabled. In this case, the first line or sentence in the doxygen comment is assumed to the the @brief line.

#### Files

Each file should start with a doxygen comment that contains the following.

/\*\*

\* @file <filename> (optional)

\* @brief A brief description (one sentence) of the purpose of the file

\*

\* Description of what the file contains and is for. This is the full

\* length description of what the brief description was saying. It can

\* be any length and multiple paragraphs.

\*

\* @author List of programmers who created and worked on this file

\* @date (This is useful if GIT fills in the date.)

\* @version <version number>

\*/

#### Classes

There should be a doxygen comment before every class that is declared in the code. Variations of this type of doxygen comment should be before any type definition such as enum declarations as well as classes.

/\*\*

\* @brief A brief description of the purpose of this definition.

\*

\* Description of class. What is its purpose and intended use.

\* Include information that a user of the class might need such as:

\*

\* - Pre-conditions needed before creating an instance of this class

\* - Post-conditions that might be needed when done using this class

\* - is the class thread-safe or not

\* - is this class a singleton in which only one instance of the class

\* may exist.

\* - What resources this class could use and which resources are not

\* shareable.

\*/

#### Variables and objects

Anytime a variable or object is defined, a doxygen comment should be used if there is any information about the it that another programmer should know. If the variable is named properly, the doxygen comment may be unnecessary. Things to keep in mind for information that should go into a doxygen comment:

* Does the value of this object effect other objects or is this object effected by other objects? In other words, does this object interact with other objects.
  + Example: If this object is true, that object values are ignored.
  + Example: This object is the lock for that object. Useful information when a set of objects are designed to be access on multiple threads.
* Is the value expressed in terms of units such as seconds, meters, meters/second?
* What conditions causes the object to change.

/\*\*

\* Description of variable or object

\*/

#### Methods

Methods, also called functions and subroutines depending on the language, are code that can be executed for other places in the program by being called. The first part of a method is the signature which includes scope (public, protected, private), the return value, the method name, a list of arguments that are passed to the method, a list of exceptions that can be thrown. This is the part of the method that comes before the body of executable statements.

The signature of a method is a contract to the caller. It is this contract that should be described in the doxygen comment just before the method. The caller does not need to know how the method performs its task. The caller does need to know what the outcome of that task is. For instance, are there any preconditions that need to be met before calling this method? What are the post-conditions that the caller should know about when the method returns? What exceptions may be thrown that the caller might want to catch and handle.

/\*\*

\* @brief Short description of method

\*

\* Long description of method.

\*

\* @param argument1 Description

\* @param argument2 Description

\* …

\* @param argumentN Description

\*

\* @return Description of return value

\*

\* @pre Description of any per-conditions that the caller is expected

\* to meet before calling this method.

\*

\* @post Description of any post-conditions the caller should expect

\* to have occurred while this method is running.

\*

\* @throws <ExceptionType1> Description on why this exception might

\* get thrown.

\* @throws <ExceptionTypeN> Description on why this exception might

\* get thrown.

\*/

#### Todo Lists

One useful feature of Doxygen is that you can generate Todo lists by using the @todo command. These doxygen comments can go anywhere in the source code files. They can be used to mark sections of code that still need to be written. Areas where the code needs extensive testing. And for any other tasks that have not been completed.

try {

… code …

} catch (Exception exc) {

/\*\*

\* @todo write error handling code for …

\*/

}

#### Other Usage

There are other times where doxygen comment can be used within the code. For example, if a section of code was written using one algorithm over another, a useful doxygen comment would list the algorithms that could have been used and why one was selected over the others. This is helpful to future programmers who might try to replace the code with code using a different algorithm that might not work as well.

### Diagrams

Doxygen can also be used to generate various diagrams useful in understanding the relationship between classes and how the program flows. The three most useful diagrams are:

* Class Diagrams which show class inheritance and which classes interact with each other.
* Call Diagrams which show what classes are used by a class.
* Caller Diagrams which show which classes used this class.

In order to generate these diagrams, Doxygen makes uses of another program, GraphViz. GraphViz may need to be installed separately.

One import configuration to enable (set to YES) when generating diagrams is the HTML\_DYNAMIC\_SECTIONS in the Doxyfile configuration file. This causes the diagrams to be hidden in the final HTML output. The diagrams can be large, covering several screens or pages. This option allows for the diagrams to be viewed by clicking on a link in the documentation. The diagrams will then be displayed inline instead of in another window.

### Doxywizard

Doxygen requires a configuration file usually called Doxyfile or .doxyfile, the name changes with operating systems. You can also specify your own name for this file. The configuration file contains many options and can be rather daunting to new users. Doxygen provides a GUI interface that can be used to both set up your configuration and run doxygen. The GUI is called doxywizard. On Linux, this program is usually run from the command line.[[1]](#footnote-2)

Doxywizard has two modes, available from tabs in the GUI. The first mode is Wizard. This should be the first mode used for configuring doxygen. It allows you to set where the source code resides, where the output documentation should go, and what diagrams should be generated.

The second mode is Expert. In this mode, all the options are presented to you. The options are grouped into common areas such as Project, HTML (output), and DOT (Diagram graphics). Once the basic configuration file is set with the Wizard, the Expert mode is useful for tweaking the configuration. One such tweak that I recommend is in the HTML section, enable HTML\_DYNAMIC\_SECTIONS.

In Expert mode, hover the mouse over an option and a description of what the option controls will appear in the lower left corner of the GUI.

Remember to save the configuration file that you generated before exiting doxywizard. Use the top menu FILE→SAVE or FILE→SAVE AS.

### Best Practices

Doxygen can be run from anywhere on your computer. It is best to create a directory or folder that Doxygen can use as a working directory. The working directory is where Doxygen stores files and temporary files. There will be a lot of files generated so you do not want this happening within your project directories.

My recommendation is to create a subdirectory or subfolder in the root directory of your project. My preference is to use the name Doc, if it is not already used by the project, or Doxy (or doxy) if Doc is already used or you want to reserve a Doc directory for later. Store your configuration file in the doxy directory.

This is the directory to use when doxywizard asks for the working directory. I use the doxy directory as the directory where doxygen should put the generated documentation. You can use a different directory for the output though you should make sure that it is directory that you can delete without losing any important files. Doxygen will create subdirectories in the output documentation directory for each time of document it generates: html, man, docbook, etc.

## Downloads

* Doxygen -- [https://www.doxygen.nl](https://www.doxygen.nl/)
  + doxywizard – part of Doxygen
* GraphViz -- <https://graphviz.org/>

1. I have not run Doxygen on Windows in over a decade and do know now if doxywizard can be run from the Start Menu or must be run from a command prompt. [↑](#footnote-ref-2)