

Doxygen Essentials

Nicholas Olsen

DOCUMENTATION. THE WORD STRIKES TERROR IN THE

programmer's soul. And for anyone who has inherited someone else's code, a lack of documentation (apart from the source itself) can make getting productive a painful chore.

Few game studios can divert precious resources exclusively to the creation and maintenance of documentation, especially when they are the sole clients for internal engine and tool development. One way to reduce the effort required for documentation is to use a documentation generator, a program that uses the source itself along with specially formatted comments to automate typesetting and page layout, which results in a final document.

Assuming interfaces will be commented (and why wouldn't they be?), using a documentation generator allows documentation to become a part of the development process rather than a process unto itself. I've found Doxygen to be a good, free solution.

GETTING STARTED WITH DOXYGEN

Doxygen, created by Dimitri van Heesch, is a documentation system for code written in C, C++, C#, Java, Objective-C, Python, IDL (CORBA and Microsoft flavors), Fortran, VHDL, PHP, and to some extent D—or so says the official literature. The commenting extensions supported by Doxygen are relatively low on programmer overhead but can scale from simple API references to comprehensive manuals.

To get started, the user must first download and install binaries for his platform, available from www.stack.nl/~fdimitri/doxygen/download.html#latestsrc. For the sake of the following examples, I will assume the user has the Doxygen executable in his PATH and available from a command-prompt, which will be the case for a default Windows installation (OS X and other *nix users may need to add Doxygen to their PATH).

The Doxywizard GUI front-end is also available to assist in configuring and running Doxygen, and is perhaps better for day-

to-day tweaking of configuration files and building documentation. I recommend playing around with Doxywizard to figure out how to use it for building the examples; descriptions of how to navigate the UI widgets would add too much clutter for a quick introduction.



YOUR FIRST DOCUMENTED CLASS

The programmer informs Doxygen of his or her intent to document something via special documentation blocks and commands. A special documentation block is a comment that is extended to inform Doxygen that it will be used for generating output.

Doxygen supports several comment styles to accommodate the various languages it understands, which can lead to confusion for beginners, as the samples in the manual bounce between styles seemingly at random. I will use C++ for the examples here, along with the triple-slash (///) to denote documentation comments and @ to denote commands.

Create a source file named DoxygenEssentials.h and insert the following comment and class declaration:

```
/// @brief A documented C++ class!
class Doxygenized {
};
```

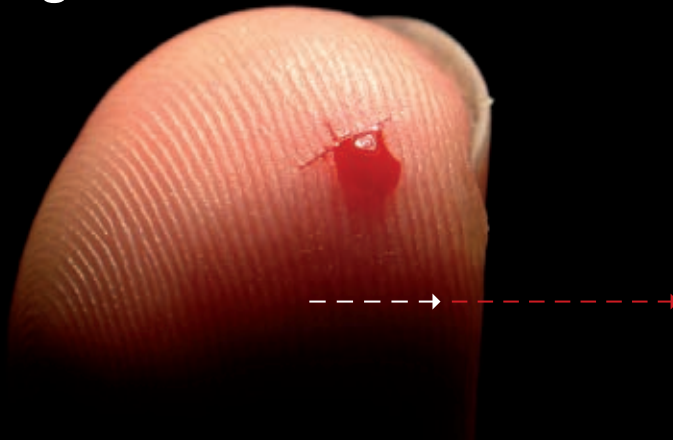
From a command prompt, cd to the directory that contains DoxygenEssentials.h and issue the following commands:

```
>doxygen -g
>doxygen
```

Your first documented class in Doxygen's Class List.

CONTINUED ON NEXT PAGE

A minor issue in the beginning.



CONTINUED FROM PREVIOUS PAGE

The first command tells Doxygen to generate a default configuration file, which will be named Doxyfile. The second command builds the documentation and places html and latex output in `./html` and `./latex`, respectively (I will ignore the LaTeX output).

Opening `./html/index.html` in a web browser will enable you to navigate to the Classes page and find the entry for Doxygenized along with the text that followed `@brief`. `@brief` is a command that instructs Doxygen to use the next paragraph of comment text as a short description for display in the class list.

ADDING A DETAILED DESCRIPTION

Now that I've given a brief description for the class list, it would be nice to have a more comprehensive description so the next poor programmer who gets stuck in this code can figure out what the class does. A detailed description can be added by using an empty documentation comment followed by a documentation block containing an arbitrary amount of descriptive text:

```
/// @brief A documented C++ class!
///
/// The Doxygenized class serves as a demonstration
/// of how to document C++ code using Doxygen
class Doxygenized {
};
```

Upon re-running Doxygen, the user can follow the Doxygenized link in the class list to a class reference page that includes the detailed description.

DOCUMENTING METHODS AND FUNCTIONS

Now that I've shown a reference page for Doxygenized, let's see what happens when a method with its own brief and detailed descriptions is added.

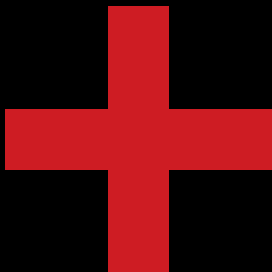
Add the following to the body of Doxygenized and rebuild:

```
public:
    /// @brief A documented method
    ///
    /// The isDocumented method serves as an
    /// example of how to document methods,
    /// along with their parameters and
    /// return value.
    bool isDocumented(int aParameter) {
        return true;
    }
```

The Doxygenized class reference now contains a Public Member Functions section where the method and brief description appear. There is also a Member Function Documentation section with a verbose entry for `isDocumented` that includes the detailed description.

In order to make the documentation more descriptive, the return type and parameter[s] can be documented explicitly, expanding the documentation block for `isDocumented` as such:

```
/// @brief A documented method
///
/// The isDocumented method serves as an
/// example of how to document methods,
/// along with their parameters and
/// return value.
/// @return Always returns true, because this
/// is a contrived example
/// @param aParameter A documented method
/// parameter. This parameter is not actually
/// used. Its only purpose in life is to be
/// documented
bool isDocumented(int aParameter) {
    return true;
}
```



Launching an MMO is a complex process. It requires skill and great fortitude, sprinkled with optimism. Each decision is dependent on the next. Before your MMO hits the crowd, the work begins to make sure your big idea is both well designed and well coded. It also has to be technically capable of coping with several thousands of concurrent players – if not, something that appears to be a minor technical issue in the beginning, can lead to a catastrophe once your game is launched.

Get Help at

www.MMOhospital.org

Because you only get one chance.

When the documentation is rebuilt, the Returns and Parameters sections can be found in `isDocumented`'s detailed description which, predictably, includes the descriptions provided for the parameter and return value.

DOCUMENTING FIELDS AND ENUMERATIONS

When documenting class fields and enumerators, the brief/detailed description pattern used in the previous examples works perfectly fine. However, Doxygen also offers shorthand for placing a brief description on the same line as a declaration.

Add the following to the body of Doxygenized:

```
/// @brief A documented enumeration!
///
/// An example of documenting an enumeration
/// and its members using the shorthand for
/// placing documentation inline with
/// declarations
enum eNumbers {
    kOne,    ///< The number One
    kTwo,    ///< The number Two
    kThree   ///< The Number Three
};

int aField; ///< An example documented field
```

Rebuilding now will produce Public Types and Public Attributes sections in the Doxygenized class reference, plus a detailed documentation section for `eNumbers` that contains the descriptions of the enumerators. This method of providing documentation on the same line as declarations keeps headers streamlined, and comes in handy when developers don't feel the need to write a treatise.

Also note that by default Doxygen only includes public and protected members in the documentation, although editing

the Doxyfile or using Doxywizard can override this behavior.

A WORD ABOUT DOXYGEN COMMANDS

Now that I've shown several Doxygen commands, using commands and their parameters will be straightforward and even obvious from here on out.

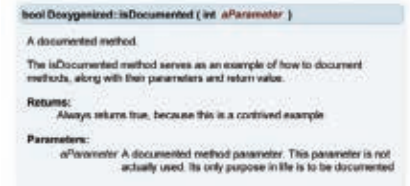
For the sake of completeness, I will briefly describe the conventions used by Doxygen's manual, which I will use when introducing additional commands.

As I've already shown, commands begin with `@` and have one or more parameters. In Doxygen's manual, the following conventions are used to describe parameters:

- If the parameter is enclosed in angle brackets ("`<parameter>`") it consumes a single word.
- If the parameter is enclosed in parenthesis ("`(parameter)`") it consumes text until the end of the line.
- If the parameter is enclosed in curly brackets ("`{parameter}`") it consumes text until the end of the next paragraph, paragraphs being delimited by a blank comment line.
- If square brackets are used as a modifier ("`[parameter]`"), the parameter is optional.

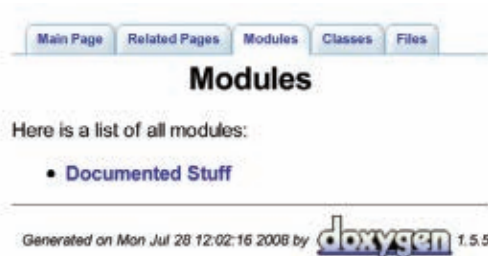
Note that these conventions only apply to how parameters are described in Doxygen's manual, not to the arguments supplied by the developer when commenting. In the preceding example, I used the `@parameter` command, which has the description `@parameter <parameter-name> { parameter description }`, to tell Doxygen which function parameter is being documented and provide a paragraph of descriptive text. Similarly, the `@return` command has the form `@return { description of return value }` and uses the paragraph that follows it as a description of possible return values.

Member Function Documentation



A detailed function description.

A minor issue can turn into a serious nightmare in the end.



A Doxygen Modules group with related classes and functions.

GROUPS

Getting back to the code, a problem with the current documentation is that class Doxygenized only appears in an alphabetical list that will grow to include every other documented class and structure. As the codebase grows, such a list will only become less useful and more uninviting.

Fortunately, Doxygen provides commands that allow classes and functions to be grouped and referenced as a unit. Wrap the existing class definition with the following documentation blocks:

```
/// @addtogroup documentedStuff Documented Stuff
/// A group used to demonstrate Doxygen's grouping
/// capabilities
/// @{
    /// [Insert the code to add to group.
    /// In this case the definition of
    /// Doxygenized].
/// @}

```

When the documentation is rebuilt, it will produce a Modules tab on the page header whose associated list includes our freshly minted group. The `@addtogroup <name> [(title)]` command takes a single-word name used to uniquely identify the group internally and create links to the group's documentation, while title is an optional descriptive title used in the output. `@addtogroup` can be used repeatedly throughout a project to add various entities in different source files, but if more than one title is provided, Doxygen uses the first one it processes.



Related Pages are automatically generated for some Doxygen commands.

CREATING A TO-DO LIST, BUG LIST, AND DEPRECATED LIST

You can use the `@todo`, `@bug`, or `@deprecated` {paragraph} commands in a documentation block to mark pieces of code as unfinished, buggy or deprecated and provide a description of the problem. Add the following definitions to DoxygenEssentials.h, inside the group created in the previous example:

```
/// @brief An unfinished class
/// @todo Figure out what this class
/// is supposed to do and implement it
class UnfinishedClass {
};

/// @brief A buggy function
/// @return num divided by zero
/// @param num number to divide by zero
/// @bug This function always crashes for some reason
int Boom(int num) {
    int divisor = 0;
    return num / divisor;
}

/// @brief A washed up, useless class
/// @deprecated Please don't ever use this code again
class Retired {
};

```

The new documentation set contains a Related Pages navigation tab with links to project-wide Todo, Bug, and Deprecated Lists. There will also be a class-specific Todo: item in the UnfinishedClass reference page, along with a Bug: in the Boom function reference and a Deprecated: item in the Retired class reference. Additionally, the Documented Stuff module now contains the new classes and function.

DECORATING FILES

Doxygen also enables the user to document files explicitly and has a few commands that lend themselves to the task. Boilerplates can be incorporated into a documentation block. Add this example file header to the top of DoxygenEssentials.h:

```
/// @file
/// @brief A documented file provided for Doxygen Essentials
///
/// Copyright 2008, Nicholas Olsen
///
/// Feel free to do whatever you want with this software
///
/// @author Nicholas Olsen
/// @version 0.0001
/// @date 2008

```

The File List entry for DoxygenEssentials.h now contains a brief comment and a link to a file reference page. This page contains entries for the classes and function in the file, while the detailed description is complete with author and date information. Note that the `@author`, `@date`, and `@version` {

description } commands can be used to document anything, but a file header is a common case where this information is used in practice.

ADDING MAIN PAGE DOCUMENTATION

While it's nice to have all this documentation, what's been created is still just a manual that is not particularly inviting to someone who does not already know where to find things. Fortunately Doxygen allows the contents of the main page to be modified, and even allows such documentation to be placed in dedicated files outside the source.

Create a file named mainpage.dox in the same directory as DoxygenEssentials.h with the following contents:

```
/// @mainpage Manual For Doxygen Essentials
///
/// @section introSection Introduction
///
/// This is the programming manual for Doxygen Essentials
///
/// @section moduleSection Modules
///
/// Doxygen Essentials contains the following modules:
/// @li @ref documentedStuff
///
/// @section additionalSection Additional Points of Interest
///
/// These may also come in handy:
/// @li @ref todo
/// @li @ref deprecated
/// @li @ref bug
```

The main page now includes Introduction, Modules, and Additional Points of Interest sections that direct readers to some of the highlights of the manual. @mainpage [(title)] instructs Doxygen to use the following documentation block as the contents of the main page, while @section <name> (title) creates a page section using name as the internal identifier and title as a descriptive heading. @li is also used to create bulleted list items that use @ref <name> ["(text)"] to link to the page or section identified by name.

WRAPPING UP

Using only brief and detailed descriptions, as outlined in this article, will result in more documentation than much of the world's code enjoys currently. By documenting parameters and return types and taking advantage of grouping commands, developers can create something that might be useful to the next programmer tasked with maintaining the code.

Add a nice main page with a fancy introduction and readers might even believe the coder knows what he is doing. Doxygen has many additional commands and capabilities that are beyond the scope of this article, the previously mentioned Doxywizard being a good place to start experimenting.

To give some examples, with the Ghostscript interpreter and a TeX distribution (I have used the TeX and Postscript tools provided by Cygwin), it's possible to create PDF reference manuals and embed LaTeX formulas in comments; install the AT&T GraphViz

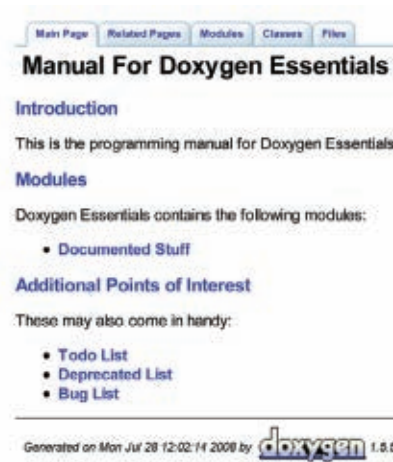
package and it becomes possible to embed Dot diagrams in your comments.

Another useful practice is to add Doxygen to an existing build or continuous integration process and publish the results on the local network, thus ensuring current documentation is always available to every developer.

Hopefully this article has inspired some game developers to consider turning their usual commenting into documenting. The next programmer who has to go through their code will thank them for it. Maybe someday you'll be pleasantly surprised to discover you wrote a manual for that code you never expected to see again.

Doxygen's home page is located at www.stack.nl/~jfdimitri/doxygen. More code examples for this article can be found at www.gdmag.com/resources/code.htm.

NICHOLAS OLSEN is an audio programmer at Double Helix Games in Irvine, California. He is a Southern California native and has been in the game industry for seven years. Contact him at nolsen@gdmag.com.



The Main Page tells readers where to find things.

www.rtpatch.com

RTPatch and Pocket Soft are registered trademarks of Pocket Soft, Inc.