DOC++

— Version 3.01 —

A Documentation System for C/C++ and Java

Roland Wunderling and Malte Zöckler

Contents

1 Introduction 4
2 Quickstart 5
3 Reference Manual 6
3.1 Usage 6
3.2 Manual Entries 8
3.3 Structure 10
3.4 File Inclusion 10
3.5 Text Formatting 10
3.5.1 supported LaTeX macors 11
3.5.2 supported HTML macros 12

4 Example — *We present a dummy class library consisting of 3 classes.* 13
4.1 CommonBase — *Common base class* 13
4.2 Intermediate — *Just to make the class graph look more interesting* 14
4.3 Derived_Class — *A derived class* 15
4.3.1 parameters 16
4.3.2 methods 17

5 Frequently Asked Questions 19
6 Distribution 21
Class Graph 22

Welcome to the wonderful world of DOC++

DOC++ is a documentation system for C/C++ and Java generating both, LaTeX output for high quality hardcopies and HTML output for sophisticated online browsing of your documentation. The documentation is extracted directly from the C++ header or Java class files.

Here is a list of highlights:

- hierarchically structured documentation
- automatic class graph generation (as Java applets for HTML)
- cross references
- high end formating support including typesetting of equations

If you think DOC++ can help you for your projects and you want to get your copy of DOC++ click on Distribution (→6, *page 21*). For an introduction to the philosophy of DOC++ go on reading the Introduction (→1, *page 4*). If you want to jump right into using DOC++ on your sources go to section Quickstart (→2, *page 5*). Section Reference Manual (→3, *page 6*) will provide you with a complete manual on the features provided DOC++. If you have problems using DOC++ you should consult our Frequently Asked Questions (→5, *page 19*). If you first want to look at some ouput of DOC++, don't do anything at all, you are just reading an example. An example for some dummy C++ classes can be found in Example (→4, *page 13*).

Enjoy!

1 Introduction

The idea of DOC++ is to provide a tool that supports the *programmer* for writing *high quality* documentation while keeping concentration on the program development. In order to do so, it is important, that the programmer can add the documentation right into the source code he/she developes. Only with such an approach, a programmer would really write some documentation to his/her classes, methods etc. and keep them up to date with upcoming changes of code. At the same time it must be possible to directly compile the code without need to a previous filter phase as needed when using e.g. cweb. Hence, the only place where to put documentation are comments.

This is exactly what DOC++ uses for generating documentation. However, there are two types of comments, the programmer wants to distinguish. One are comments he/she does for remembering some implementational issues, while the others are comments for documenting classes, functions etc. such that he/she or someone else would be able to use the code later on. In DOC++ this distingtion is done via different types of comments. Similar to JavaDoc, documentation comments are of the following format

- `/** ... */`
- `/// ...`

where the documentation is given in `...`. Such comments are referred to as *DOC++ comments*. Each DOC++ comment generates a manual entry for the *next* declaration in the source code.

Now, let's consider what "high quality" documentation means. Many programmers like to view the documentation online simply by klicking their mouse buttons. A standard for such documents is HTML for which good viewers are available on almost every machine. Hence, DOC++ has been designed to produce HTML output in a structured way.

But have you ever printed a HTML-page? Doesn't it look ugly, compared to what one is used to. This is not a problem for DOC++ since it also provides LaTeX output for generating high quality hardcopies.

For both ouput formats, it is important that the documentation is well structured. DOC++ provides hierarchies, that are refelected as sections/subsection etc. or HTML-page hierarchies, respectively. Also an index is generated that allows the user to easily find what he/she looks for.

As C++ and Java are (somewhat) object-oriented languages, another type of hierarchy is introduced, namely class hierarchies. The best way to read such a hierarchy is by looking at a picture of it. Indeed, DOC++ automatically draws a picture for each class derivation hierarchy or shows it with a Java applet in the HTML output.

An additional goody of DOC++ is its ability to generate a LaTex typeset source code listing for C/C++ code.

2 Quickstart

If you want to jump straight into using DOC++, add a line like

```
/// ...
```

before each function, variable, class, define, etc. you wish to document. For "..." you may choose to add a short documentation string for the entry. You will typically want to do so for your header files only. If you intend to write more than one line of documentation, succeed this line with a comment like

```
/**
   ...
 */
```

and put the long documentation text in place of "...". A source file containing such comments is said to be *docified*. You may call

```
> docify <original> <docfile>
```

from your shell to create a docified copy <docfile> form your <original> file. The > indicates the shell prompt and must not to be typed.

Now start DOC++ by typing:

```
> doc++ -d html <files>
```

for HTML output or

```
> doc++ -t -o doc.tex <files>
```

for LaTeX output in you shell, where <files> is the list of docified files.

Each `///` -comment yields one manual entry. If you need to group manual entries, you may do so with the construction

```
/**@name <name for the group>
 *  <documentation for the group>
 */
//@{
   <other manual entries>
//@}
```

This will create an entry with the specified name, that contains all <other manual entries> as subentries. Note however, that class members are automatically set as subentries of the class's manual entry. You also may include other files using the comment

```
//@Include: <file(s)>
```

3 Reference Manual

Names

3.1	**Usage**	6
3.2	**Manual Entries**	8
3.3	**Structure**	10
3.4	**File Inclusion**	10
3.5	**Text Formatting**	10

DOC++ follows the approach of maintaining one source code that contains both, the C++ or Java program itself along with the documentation in order to avoid incompatibilities between the programm and its documentation. Unlike other approaches such as WEB, sources documented with DOC++ are ready to be compiled without need of any preprocessing (like tangle). We feel, that this is of great advantage for intensive programming and debugging activities.

This documentation is organized as follows. Section Usage (→3.1, *page 6*) describes how to generate your documentation for readily docified sources, hence explains the comand line options of DOC++. Section Manual Entries (→3.2, *page 8*) discusses, how the manual entries generated for DOC++ comments are built up and section Structure (→3.3, *page 10*) how to structure your documentation hierarchically. Finally section Text Formatting (→3.5, *page 10*) describes all the features provided by DOC++ to format the documentation text (such as bold face typesetting etc.).

3.1 Usage

Calling DOC++ with option `-h` will give you a long screen with one-line descriptions of the command line options provided by DOC++. However, we now attempt to provide a more detailed description suitable for a novice user to understand how to call DOC++ with his docified sources.

At the commandline DOC++ may be called with a sequence of options and a list of files or directories. No option may be passed after the first filename. All files passed to DOC++ are parsed in the order they are specified for generating documentation from them. All directories are traversed recursively and all files `*.h*` or `*.java` (depending on the -J comand line option) are parsed. However, it is good practice to control the input files with one main input file and use the `@Include:` directive.

Options consist of a leading character `-`, preceeded by one or two characters and optionally a space-separated argument.

Command line options come in three different flavours. The first type of options control parameters that are independent of the chosen output, HTML (the default) or LaTeX (selected with option `-t`). These are:

-h This instructs DOC++ to print a one-line description of all options to the screen.

-J Sets DOC++ into Java mode, i.e. lets DOC++ parse Java instead of C/C++.

-A Instructs DOC++ to generate manual entries for every declaration it finds, no matter if it is documented with a DOC++ comment or not. instead of C/C++.

-p instructs DOC++ also to include private class members to the documentation. If not specified no private member will show up in the documentation (even if they are docified).

-v Sets DOC++ into verbose mode making it operate more noisy. This may be helpful when debugging your documentation.

-u Draw arrows from derived class to the base class when generating class graphs.

-k Also generate class 'graphs' for classes with neither base class nor child classes.

-H Instructs DOC++ to use HTML as formatting language.

The following command line options are only active when HTML output is selected, i.e. no option `-t` is passed:

-d <name> This specifies the directory <name>, where the HTML files and gifs are to be written. If not specified, the current directory will be used. If the specified directory does not yet exist, it will be created by DOC++.

-f instructs DOC++ to write on each HTML page the file of the source code, where this manual entry has been declared.

-g instructs DOC++ not to generate gifs for equations and `\TEX{}` text in the documentation. This may reduce execution time when calling DOC++, but note, that DOC++ keeps a database of already generated gifs, such that gifs are not recreated if they already exist. However, if you do not have LaTeX, dvips, ghostscript and the ppmtools installed on your system, you *must* use this option, since then DOC++ will fail setting up the gifs.

-G This instructs DOC++ to reconstruct all gifs, even if they already exist. This may be useful, if the database is corrupted for some reason.

-B <file> will add <file> as the bottom banner to every HTML page generated by DOC++. This is how to get rid of DOC++ logos and customize the output for your needs.

-a When this option is specified, DOC++ will use HTML tables for listing the members of a class. This yields all member names to be aligned.

-b Same as `-a` except that a bordered table will be used.

-r This option should only be set when using DOC++ on DOS filesystems with 8-character-long filenames. DOC++ will respect this (stupid) convention.

-j Suppresses the generation of java applets for drawing class graphs.

Finally this set of command line options provides some control for the LaTeX output of DOC++.

-t Instructs DOC++ to produce LaTeX output rather than HTML.

-o <file> Sets the output file name. If not specified, the output is printed to stdout.

-l Switches off the generation of the LaTeX environment. This should be used if you intend to include the documentation in some LaTeX document.

-eo <option> adds <option> to LaTeX's `\documentclass`.

-ep <package> adds `\usepackage{package}` to the LaTeX environment.

-ep <file> uses the contents of <file> as LaTeX environment.

-s Instead of generating a manual from the manual entries, DOC++ will generate a source code listing. This listing contains all *normal* C or C++ comments typeset in LaTeX quality. Every line is preceeded with its line number.

For customization of the LaTeX output, please try to understand and edit the style file `docxx.sty`. (Sorry, there is no documentation on how to do this.) The HTML output can be customized by means of the following 6 files:

`indexHeader.inc` Header for index HTML-pages

`indexFooter.inc` Footer for index HTML-pages

`hierHeader.inc` Header for class hierarchy HTML-pages

`hierFooter.inc` Footer for class hierarchy HTML-pages

`classHeader.inc` Header for all other HTML-pages

`classFooter.inc` Footer for all other HTML-pages

If one or more of these files are found in the current directory, the corresponding part of a HTML-page is substituted by the contents of the file. The files `indexHeader.inc` and `hierHeader.inc` should start with `<HTML><TITLE> ...`. File `classHeader.inc` should start with `<BODY> ...`, since for such pages DOC++ sets up the title.

As an example, the LaTeX version of this document has been generated with

```
doc++ -v -t -o doc.tex -ep a4wide doc.dxx
```

while the HTML version has been created using

```
doc++ doc.dxx
```

As you can see, this documentation itself is written using DOC++ in order to gain the benefits of having just one documentation source and two different output possibilities.

3.2 Manual Entries

Just like in JavaDoc the documentation for DOC++ is contained in special versions of Java, C or C++ comment. These are comments with the format:

```
/**
  *  ...
  * /
```

Note, that DOC++ comments are only those with a double asterisk `/**`. We shall refer to such a comment as a DOC++ comment. Each DOC++ comment is used to specify the documentation for the *subsequent* declaration (of a variable, class, etc.).

Every DOC++ comment defines a *manual entry*. A manual entry consists of the documentation provided in the DOC++ comment and some information from the subsequent declaration, if available.

Manual entries are structured into various *fields*. Some of them are automatically filled in by DOC++ while the others may be specified by the documentation writer. Here is the list of the fields of manual entries:

Field name	provider	description
`@type`	DOC++	depends on source code
`@name`	both	depends on source code
`@args`	DOC++	depends on source code
`@memo`	user	short documentation
`@doc`	user	long documentation
`@return`	user	doc of return value of a function
`@param`	user	doc of parameter of a function
`@exception`	user	doc for exepction thrown by a function
`@see`	user	cross reference
`@author`	user	author
`@version`	user	version

Except for explicite manual entries, the first three fields will generally be filled automatically by DOC++. How they are filled depends on the *category* of a manual entry, which is determined by the source code following a DOC++ comment. Generally they contain the entire signature of the subsequent declaration. The following table lists all categories of manual entries and how the fields are filled

Category	`@type`	`@name`	`@args`
macro	#define	name	[argument list]
variable	Type	name	-
function/method	Return type	name	arguments list [exceptions]
union/enum	union/enum	name	-
class/struct	class/struct	name	[derived classes]
interface	interface	name	[extended interfaces]

In any case `@name` contains the name of the declaration to be documented. It will be included in the table of contents.

The remaining fields are filled from the text in the DOC++ comment. Except for the `@doc` and `@memo` field, the text for a field must be preceeded by the field name in the beginning of a line of the DOC++ comment. The subsequent text up to the next occurrence of a field name is used for the field. Field `@name` is an exception in that only the remaining text in the same line is used to fill the field. As an example

```
@author Roland Wunderling and Malte Z\"ockler
```

is used to fill the `@author` field with the text "Roland Wunderling and Malte Zöckler".

Text that is not preceeded by a field name is used for the `@doc` field. The very first text in a DOC++ comment up to the first occurrence of character '.' is also copied to the `@memo` field. This may be overridden by explicitly specifying a `@memo` field. In this case also characters '.' are allowed.

The fields `@type`, `@args` and `@doc` may not be filled explicitely.

3.3 Structure

DOC++ automatically imposes a hierarchical stucture to the manual entries for classes, structs, unions, enums and interfaces, in that it organizes members of such as sub-entries.

Additionally DOC++ provides means for manually creating subentries to a manual entry. This is done via *documentation scopes*. A documentation scope is defined using a pair of brackets:

```
//@{
    ...
//@}
```

just like variable scopes in C/C++ or Java. Instead of `//@{` and `//@}` one can also use `/*@{*/` and `/*@}*/`. All the manual entries within a documentation scope are organized as subentries of the manual entry preceeding the opening bracket of the scope, but only if this is an explicit manual entry. Otherwise a dummy explicite manual entry is created.

In addition to this Java allows the programmer to organize classes hierarchically by means of `packages`. Packages are directly represented in the manual entry hierarchy generated by DOC++. When a DOC++ comment is found before a `package` statement, the documentation is added to the package's manual entry. This functionality as well as documentation scopes are extensions to the features of JavaDoc.

3.4 File Inclusion

There is one more special type of comments for DOC++, namely `//@Include: <files>` or `/*@Include: <files>*/`. When any of such comments is parsed, DOC++ will read the specified files in the order they are given. Also wildcards using * are allowed. It is good practice to use one input file only and include all documented files using such comments, especially when explicite manual entries are used for structuring the documentation. This text is a good example for such a documentation.

3.5 Text Formatting

Names

3.5.1	**supported LaTeX macors**	11
3.5.2	**supported HTML macros**	12

DOC++ provides both, HTML and LaTeX output. Both languages have formatting macros which are more or less powerful. The idea of DOC++ is to be able to generate both ouput formats from a single source. Hence, it is not possible to rely on the full functionality of either formatting macros. Instead, DOC++ supports a subset of each set of macros, that has proved to suffice for most applications. However, in one run of DOC++ the user must decide for the formating macros to use. The subset of each macro packet is listed in the following subsections. If one uses only one of the subsets, goodlooking output can be expected for both formats.

3.5.1

supported LaTeX macors

This is the subset of LaTeX formatting instructions provided by DOC++:

$...$ math mode for inline equations (example $\sqrt{\frac{x^2}{2}}$).

\[...\] display math mode

\# to output character #

_ to output character "_"

**\ ** to output character " "

\em only to be used as `{\em ...}`

\bf only to be used as `{\bf ...}`

\it only to be used as `{\it ...}`

\tt only to be used as `{\tt ...}`

\tiny only to be used as `{\tiny ...}`

\scriptsize only to be used as `{\scriptsize ...}`

\footnotesize only to be used as `{\footnotesize ...}`

\small only to be used as `{\small ...}`

\large only to be used as `{\large ...}`

\Large only to be used as `{\Large ...}`

\LARGE only to be used as `{\LARGE ...}`

\huge only to be used as `{\huge ...}`

\Huge only to be used as `{\Huge ...}`

\HUGE only to be used as `{\HUGE ...}`

center i.e. `\begin{center} ... \end{center}`

flushleft i.e. `\begin{flushleft} ... \end{flushleft}`

flushright i.e. `\begin{flushright} ... \end{flushright}`

verbatim i.e. `\begin{verbatim} ... \end{verbatim}`

tabular i.e. `\begin{tabular}{lll} ...&...\\ ... \end{tabular}`

itemize i.e. `\begin{itemize} \item ... ... \end{itemize}`

enumerate i.e. `\begin{enumerate} \item ... ... \end{enumerate}`

description i.e. `\begin{description} \item[...] ... ... \end{description}`

When writing your documentation using only this, you can be sure to get reasonable LaTeX **and** HTML documentation for your code.

There are some additional LaTeX macros provided by DOC++:

- `#...#` corresponds to the LaTeX `\verb!...!`, i.e. outputs `...` verbatim.
- `\Ref{...}` allows to specify a reference to a manual entry with name `...`.
- `\URL{...}` allows to specify a link to the WWW page `...`.
- `\URL[my text]{...}` allows to specify `my text` as the links name.
- `\TEX{...}` allows to include any complicated LaTeX code into you document. For HTML output DOC++ will have LaTeX to process it, produce gifs out of it and includes them into the HTML document. NOTE: This requires LaTeX, dvips, ghostscript and ppmtools to be correctly installed to your system!

3.5.2 supported HTML macros

This is the subset of HTML formatting instructions provided by DOC++:

`<p>` paragraph

`<em>` **...** `<\em>` emphasize

`<i>` **...** `<\i>` italic

`<b>` **...** `<\b>` bold face

verbatim i.e. `<pre>` ... `</pre>`

description i.e. `<dl>` `<dt>`...`<dd>` ... `</dl>`

itemize i.e. `<ul>` `<li>` ... `</ul>`

enumerations i.e. `<ll>` `<li>` ... `</ll>`

4 Example

We present a dummy class library consisting of 3 classes.

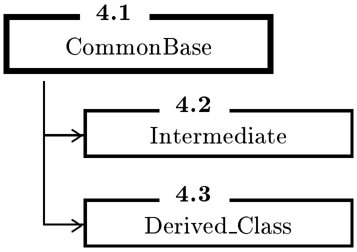
Names

4.1	class	**CommonBase**	*Common base class*	13
4.2	class	**Intermediate** : public CommonBase, public NotDocified	*Just to make the class graph look more interesting*	14
4.3	class	**Derived_Class** : public CommonBase, protected Intermediate	*A derived class*	15
4.4	int	**function** (const DerivedClass& c)	*A global function*	17

4.1 class **CommonBase**

Common base class

Inheritance



Public Members

const Derived_Class&
getB (const Intermediate& c)
a public member function showing links to argument and type classes

Protected Members

double	**variable**	*a protected member variable*

Common base class. This could be a long documentation for the common base class. Note that protected members are displayed after public ones, even if they appear in another order in the source code.

This is how this documentation has been generated:

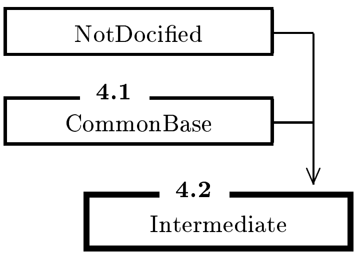
```
/** Common base class.
    This could be a long documentation for the common base class.
    Note that protected members are displayed after public ones, even if they
    appear in another order in the source code.
    This is how this documentation has been generated: * /

class CommonBase
{
private:
        /// this member shows up only, if you call doc++ with -p
    int privateMember() ;

protected:
        /// a protected member variable
    double variable ;

public:
        /// a public member function showing links to argument and type classes
    const Derived_Class& getB( const Intermediate& c ) const ;
} ;
```

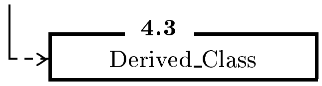
4.2

class **Intermediate** : public CommonBase, public NotDocified

Just to make the class graph look more interesting

Inheritance





Just to make the class graph look more interesting. Here we show multiple inheritance from one docified class and a nondocified one.

This is how this documentation has been generated:

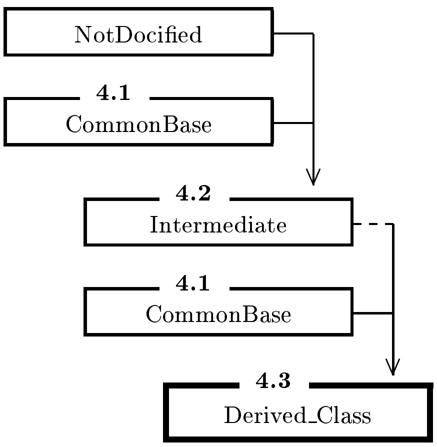
```
/** Just to make the class graph look more interesting.
    Here we show multiple inheritance from one docified class and a nondocified
    one.

    This is how this documentation has been generated: * /

class Intermediate : public CommonBase, public NotDocified
{
} ;
```

4.3

class **Derived_Class** : public CommonBase, protected Intermediate

A derived class

Inheritance



Public Members

4.3.1	**parameters**	16
4.3.2	**methods**	17

A derived class. Here we show multiple inheritance from two docified classes. This example shows how to structure the members of a class, if desired.

This is how this documentation has been generated:

```
/** A derived class.
    Here we show multiple inheritance from two docified classes.
    This example shows how to structure the members of a class, if desired.

    This is how this documentation has been generated: * /

class  Derived_Class : public CommonBase, protected Intermediate
{
public:
    /**@name parameters * /
    //@{
        /// the first parameter
    double a ;
        /// a second parameter
    int b ;
    //@}

    /**@name methods * /
    //@{
        /// constructor
        /** This constructor takes two arguments, just for the sake of
            demonstrating how documented members are displayed by DOC++.
            @param  a this is good for many things
            @param  b this is good for nothing  * /
    DerivedClass( double a, int b ) ;
        /// destructor
    ~DerivedClass() ;
    //@}
} ;
```

4.3.1

parameters

Names

double	**a**	*the first parameter*
int	**b**	*a second parameter*

4.3.2

methods

Names

4.3.2.1 **Derived_Class** (double a, int b)
Constructor 17

~Derived_Class () *destructor*

4.3.2.1

Derived_Class (double a, int b)

Constructor

Constructor. This constructor takes two arguments, just for the sake of demonstrating how documented members are displayed by DOC++.

Parameters: `a` — this is good for many things
`b` — this is good for nothing

4.4

int **function** (const DerivedClass& c)

A global function

A global function. As promised, not only classes and members can be documented with DOC++. This is an example for how to document global scope functions. You'll notice, that there is no technical difference to documenting member functions. The same applies to variables or macros.

This is how this documentation has been generated:

```
/** A global function.
    As promised, not only classes and members can be documented with DOC++.
    This is an example for how to document global scope functions.
    You'll
    notice, that there is no technical difference to documenting member
    functions. The same applies to variables or macros.

    @param c reference to input data object
    @return whatever
    @author Snoopy
    @version 3.00 beta
```

```
@see DerivedClass * /

int function( const DerivedClass& c ) ;
```

Return Value:	whatever
Parameters:	`c` — reference to input data object
See Also:	DerivedClass
Author:	Snoopy
Version:	3.00 beta

5 Frequently Asked Questions

Q: How can I group a number of entries ?

A: Right as in this example:

```
/**@name comparison operators * /
//@{
  /// equal
  bool operator==(const Date& cmpDate);
  ///
  bool operator!=(const Date& cmpDate);
  /// less
  bool operator<(const Date& cmpDate);
  /// greater
  bool operator>(const Date& cmpDate);
//@}
```

Q: How can I influence the order of the entries ?

A: The order of class members is the same as in the class declaration. The order of the entries in the table of contents is the order in which doc++ reads the classes. Hence, typing "doc++ *" yields an alphabetically ordered list. You may also use `//@Include:` to read your files in the desired order.

Q: How can I change fonts/borders/whatever in LaTeX ?

A: Edit the File docxx.sty (there is no documentation about how to do this, sorry :-( ).

Q: What do the blue and grey balls in the HTML-output mean ?

A: Entries that have a doc-string (not only memo) have a blue ball. Clicking on this ball gets you to the documentation.

Q: How can I avoid scrolling all the way down to the class' documentation?

A: Click on the books (on the left of the classname) to jump there.

Q: How can I get other paper formats for the LaTeX output?

A: Try the -e. options. E.g.: with -eo a4paper, the a4paper option will be set for the documentstyle; with -ep a4wide a `\usepackage{a4wide}` will be inserted before `\begin{document}`. Finally, one can provide a completely own LaTeX environment setup using the -ef option.

Q: I have the following:

```
///
    class A { ... } a;
```

Why do I get scrambled results ?

A: DOC++ does not know what you intend to document, the class A or the variable a. Solution: Split up class and variable declarations like this:

```
    ///
class A { ... };
    ///
A a;
```

Q: I have the following old C typedef:
`/** ... */`
`typedef struct a { ... } a_t ;`
Why do I get scrambled results ?

A: This is the same problem as above. The solution is also equivalent:
`/** ... */`
`struct a { ... };`
`/** ... */`
`typedef struct a a_t ;`

Q: Is there a way to make the equation font larger in the HTML output?

A: Sure, more than one. You may use `\large` or so within the equations. Or you may use the option -eo 12pt to render all gifs in 12pt instead of 10pt. Or you may use you own latex environment with -ef to setup all fonts as desired.

Q: Why does doc++ fail to build gifs for my formulae?

A: There are two typical kinds of failour. One is, that you don't have set up you path to find the ppmtools, gs or latex. The other is, that latex fails to process your formulae. Check the file dxxgifs.tex in your html directory to see what latex tries to process.

6 Distribution

DOC++ comes readily compiled for various platforms. Please browse our `distributions` directory to find out if your platform is already available. If not, you will have to compile DOC++ by yourself. Here (`docxxsrc.tar.gz`) are the sources for you to do so (or as zip-file (`docxxsrc.zip`)).

The documentation is available here (`docxxdoc.tar.gz`) (or as zip-file (`docxxdoc.zip`)), or you may prefer directly accessing a postscript version here (`doc.ps`). Note however that you can create this documentation by yourself after installing DOC++ to your system.

Note, that in order to take advantage of all features of DOC++, you will need a complete and correct installation of the following packages:

- LaTeX2e (`http://www.cl.cam.ac.uk/CTAN/latex/ftp.html`) (including dvips)
- ghostscript (`ftp://ftp.cs.wisc.edu/ghost/gnu`)
- ppmtools (`ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM/`)

Follow the links to get them.

Class Graph

- **4.1** CommonBase 13
  - **4.2** Intermediate 14
    - **4.3** Derived_Class 15
  - **4.3** Derived_Class 15