1 {@link http://phpdocu.sourceforge.net phpDocumentor 1.2.2} Tutorial

{@link http://phpdocu.sourceforge.net phpDocumentor 1.2.2} Tutorial — learn how to use phpDocumentor to document, and its internals

Synopsis

{@toc}

Introduction

phpDocumentor is the most advanced automatic documentation system written for PHP, in PHP. This package has many features:

- *New* Exciting new tokenizer-based parser is literally twice as fast as previous parser!
- *New* {@tutorial tags.category.pkg} tag, useful for peardoc2 and other uses.
- *New* new CSV:dia2code Converter generates output that can be used to create a UML diagram, and generate code from the diagram using Harald Fielkers {@link http://sf.net/projects/dia2code dia2code project}
- *New* versatile {@tutorial tags.internal.pkg}, {@tutorial tags.inlineinternal.pkg}, {@tutorial tags.example.pkg} tags
- *New* Brand new extensive phpDocumentor manual
- *New* extended/tutorial documentation in docbook format with linking to any element and to other documentation, including sub-sections, from the source code (see {@tutorial tutorials.pkg})
- *New* docblock templates to cut down on repetition
- *New* {@}source} inline tag to display function source
- *New* XML:DocBook:peardoc2 templates for PEAR developers
- *New* Greater ease of extending a Converter, see {@tutorial Converters/Converters.pkg}
- *New* Options are moved from source code to phpDocumentor.ini, see {@tutorial phpDocumentor.howto.pkg#using.phpdocumentorini}
- ability to parse any PHP file, regardless of documentation format
- conforms loosely to the {@link http://java.sun.com/docs/books/jls/first_edition/html/18.doc.html JavaDOC protocol}, and will be familiar to Java programmers
- documents all includes, constants, functions, static functions, classes, methods, static variables, class variables, and can document global variables and external tutorials
- auto-linking to pre-defined PHP functions
- Output in HTML, CHM, PDF, XML DocBook formats
- templateable with many bundled templates
- automatic linking to elements in any documented package
- documents name conflicts between packages to help avoid PHP errors
- documentation CVS directly.
- support for JavaDoc doclet-like output through Converters
- error/warning tracking system
- extreme class intelligence: inherits documentation, package

- complete phpdoc.de DocBlock tag support. Additions include @var, @magic, @deprec, @todo, and phpdoc.de parsing of @param.
- alphabetical indexing of all elements by package and overall
- class trees
- MUCH more than just this short list

phpDocumentor Basics

Starting Out From Scratch

The documentation process begins with the most basic element of phpDocumentor: a *Documentation block* or *DocBlock*. A basic DocBlock looks like this:

```
/**
 *
 */
```

A DocBlock is an extended C++-style PHP comment that begins with "/**" and has an "*" at the beginning of every line. DocBlocks precede the element they are documenting.

Caution Any line within a DocBlock that doesn't begin with a * will be ignored.

To document function "foo()", place the DocBlock immediately before the function declaration:

```
/**
 * Defies imagination, extends boundaries and saves the world ...all before breakfast!
 */
function foo()
{
}
```

This example will apply the DocBlock to "define('me',2);" instead of to "function foo()":

```
/**
 * DocBlock for function foo?
 *
 * No, this will be for the constant me!
 */
define('me',2);
function foo($param = me)
{
}
```

define() statements, functions, classes, class methods, and class vars, include() statements, and global variables can all be documented, see {@tutorial phpDocumentor.howto.pkg#documenting.elements}.

DocBlocks

A DocBlock contains three basic segments in this order:

- Short Description
- Long Description
- Tags

The Short Description starts on the first line, and can be terminated with a blank line or a period. A period inside a word (like example.com or 0.1 %) is ignored. If the Short Description would become more than three lines long, only the first line is taken. The Long Description continues for as many lines as desired and may contain html markup for display formatting. Here is a sample DocBlock with a Short and a Long Description:

```
/**
 * return the date of Easter
 *
 * Using the formula from "Formulas that are way too complicated for anyone to
 * ever understand except for me" by Irwin Nerdy, this function calculates the
 * date of Easter given a date in the Ancient Mayan Calendar, if you can also
 * guess the birthday of the author.
 */
```

Optionally, you may enclose all paragraphs in a tag. Be careful, if the first paragraph does not begin with , phpDocumentor will assume that the DocBlock is using the simple double linebreak to define paragraph breaks as in:

```
/**
 * Short desc
 *
 * Long description first sentence starts here
 * and continues on this line for a while
 * finally concluding here at the end of
 * this paragraph
 *
 * The blank line above denotes a paragraph break
 */
```

Here is an example of using

```
/**
 * Short desc
 *
 * <p>Long description first sentence starts here
 * and continues on this line for a while
 * finally concluding here at the end of
 * this paragraph</p>
 * This text is completely ignored! it is not enclosed in p tags
 * <p>This is a new paragraph</p>
 */
```

phpDocumentor also supports JavaDoc's DocBlock format through the command-line option {@tutorial phpDocumentor.howto.pkg#usin line.javadocdesc}. Due to the non-xhtml compliant unmatched p tag, we highly recommend you avoid this syntax whenever possible

```
/**
 * <p>
 * Short desc is only to the first period.
 * This means a sentence like:
 * "Parses Mr./Mrs. out of $_GET." will
 * parse a short description of "Parses Mr."
 * which is rather silly.  Long description is
 * the entire DocBlock description including the
 * Short desc, and paragraphs begin where p is like:
 * <p>
 * The p above denotes a paragraph break
 */
```

phpDocumentor will convert all whitespace into a single space in the long description, use paragraph breaks to define newlines, or <pre>, as discussed in the next section

DocBlock Description details

As of phpDocumentor 1.2.0, the long and short description of a DocBlock is parsed for a few select html tags that determine additional formatting. Due to the nature of phpDocumentor's output as multiple-format, straight html is not allowed in a DocBlock, and will be converted into plain text by all of the converters unless it is one of these tags:

- <b> -- emphasize/bold text
- <code> -- Use this to surround php code, some converters will highlight it
-
 -- hard line break, may be ignored by some converters
- <i> -- italicize/mark as important
- <kbd> -- denote keyboard input/screen display
- -- list item
- -- ordered list
- <p> -- If used to enclose all paragraphs, otherwise it will be considered text
- <pre> -- Preserve line breaks and spacing, and assume all tags are text (like XML's CDATA)
- <samp> -- denote sample or examples (non-php)
- -- unordered list
- <var> -- denote a variable name

Do not think of these tags as text, they are parsed into objects and converted into the appropriate output format by the converter. So, a b tag may become <emphasis> in DocBook, and a <p> tag might become " " (4 spaces) in the PDF converter! The text output is determined by the template options file options.ini found in the base directory of every template. For instance, the options.ini file for the HTML:frames:default template is in phpDocumentor/Converters/HTML/frames/templates/default/options.ini

For the rare case when the text "<b>" is needed in a DocBlock, use a double delimiter as in <<b>>. phpDocumentor will automatically translate that to the physical text "<b>".

Using <code> and <pre> Both <code> and <pre> ignore any html listed above except for their closing tags </code> for <code> and </pre> for <pre>

New 1.2.0rc1: If you need to use the closing comment "*/" in a DocBlock, use the special escape sequence "{@*}." Here's an example:

```
/**
 * Simple DocBlock with a code example containing a docblock
 *
 * <code>
 *  /**
 *   * My sample DocBlock in code
 *   {@}*}
 * </code>
 */
```

This will parse as if it were:

```
/**
 * Simple DocBlock with a code example containing a docblock
 *
 * <code>
 *  /**
 *   * My sample DocBlock in code
 *   */
 * </code>
 */
```

New 1.2.0rc1: The phpEdit tool supports very clever list extraction from DocBlocks, and now phpDocumentor supports the same cleverness. This example:

```
/**
 * Simple DocBlock with simple lists
 *
 * Here's a simple list:
 * - item 1
 * - item 2, this one
 *   is multi-line
 * - item 3
 * end of list.  Next list is ordered
 * 1 ordered item 1
 * 2 ordered item 2
 * end of list. This is also ordered:
 * 1. ordered item 1
 * 2. ordered item 2
 */
```

phpDocumentor recognizes any simple list that begins with "-", "+", "#" and "o", and any ordered list with sequential numbers or numbers followed by "." as above. The list delimiter must be followed by a space (not a tab!) and whitespace must line up exactly, or phpDocumentor will not recognize the list items as being in the same list.

```
/**
 * Simple DocBlock with screwy lists
 *
 * +not a list at all, no space
 * Here's 3 lists!
 * - item 1
 *  - item 2, this one
```

```
 *   is multi-line
 *- item 3
 */
```

For in-depth information on how phpDocumentor parses the description field, see {@link ParserDescCleanup.inc}

DocBlock Templates

New for version 1.2.0, phpDocumentor supports the use of DocBlock templates. The purpose of a DocBlock template is to reduce redundant typing. For instance, if a large number of class variables are private, one would use a DocBlock template to mark them as private. DocBlock templates simply augment any normal DocBlocks found in the template block.

A DocBlock template is distinguished from a normal DocBlock by its header. Here is the most basic DocBlock template:

```
/**#@+
 *
 */
```

The text that marks this as a DocBlock template is "/**#@+" - all 6 characters must be present. DocBlock templates are applied to all documentable elements until the ending template marker:

```
/**#@-*/
```

Note that all 8 characters must appear as "/**#@-*/" in order for phpDocumentor to recognize them as a template. Here is an example of a DocBlock template in action:

```
class Bob
{
    // beginning of docblock template area
    /**#@+
     * @access private
     * @var string
     */
    var $_var1 = 'hello';
    var $_var2 = 'my';
    var $_var3 = 'name';
    var $_var4 = 'is';
    var $_var5 = 'Bob';
    var $_var6 = 'and';
    var $_var7 = 'I';
    /**
     * Two words
     */
    var $_var8 = 'like strings';
    /**#@-*/
    var $publicvar = 'Lookee me!';
}
```

This example will parse as if it were:

```
class Bob
{
    // beginning of docblock template area
```

```
    /**
     * @access private
     * @var string
     */
    var $_var1 = 'hello';
    /**
     * @access private
     * @var string
     */
    var $_var2 = 'my';
    /**
     * @access private
     * @var string
     */
    var $_var3 = 'name';
    /**
     * @access private
     * @var string
     */
    var $_var4 = 'is';
    /**
     * @access private
     * @var string
     */
    var $_var5 = 'Bob';
    /**
     * @access private
     * @var string
     */
    var $_var6 = 'and';
    /**
     * @access private
     * @var string
     */
    var $_var7 = 'I';
    /**
     * Two words
     * @access private
     * @var string
     */
    var $_var8 = 'like strings';
    var $publicvar = 'Lookee me!';
}
```

Note that for `$_var8` the DocBlock template merged with the DocBlock. The rules for merging are simple:

- The long description of the docblock template is added to the front of the long description. The short description is ignored.
- All tags are merged from the docblock template

Tags

Tags are single words prefixed by a "@" symbol. Tags inform phpDocumentor how to present information and modify display of documentation. All tags are optional, but if you use a tag, they do have specific requirements to parse properly.

A list of all possible tags in phpDocumentor follows:

```
/**
 * The short description
```

```
 *
 * As many lines of extendend description as you want {@}link element} links to an element
 * {@}link http://www.example.com Example hyperlink inline link} links to a website. The ↩
    inline
 * source tag displays function source code in the description:
 * {@}source}
 *
 * In addition, in version 1.2+ one can link to extended documentation like this
 * documentation using {@}tutorial phpDocumentor/phpDocumentor.howto.pkg}
 * In a method/class var, {@}inheritdoc may be used to copy documentation from
 * the parent method
 * {@internal
 * This paragraph explains very detailed information that will only
 * be of use to advanced developers, and can contain
 * {@}link http://www.example.com Other inline links!} as well as text}}
 *
 * Here are the tags:
 *
 * @abstract
 * @access       public or private
 * @author       author name <author@email>
 * @copyright    name date
 * @deprecated   description
 * @deprec       alias for deprecated
 * @example      /path/to/example
 * @exception    Javadoc-compatible, use as needed
 * @global       type $globalvarname
   or
 * @global       type description of global variable usage in a function
 * @ignore
 * @internal     private information for advanced developers only
 * @param        type [$varname] description
 * @return       type description
 * @link         URL
 * @name         procpagealias
   or
 * @name         $globalvaralias
 * @magic        phpdoc.de compatibility
 * @package      package name
 * @see          name of another element that can be documented, produces a link to it in ↩
    the documentation
 * @since        a version or a date
 * @static
 * @staticvar    type description of static variable usage in a function
 * @subpackage  sub package name, groupings inside of a project
 * @throws       Javadoc-compatible, use as needed
 * @todo         phpdoc.de compatibility
 * @var   type  a data type for a class variable
 * @version version
 */
function if_there_is_an_inline_source_tag_this_must_be_a_function()
{
...
}
```

In addition, tutorials allow two addition inline tags: {@}id}, used to allow direct linking to sections in a tutorial, and {@}toc}, used to generate a table of contents from {@}id}s in the file. Think of {@}id} like an <a name="idname"> HTML tag, it serves the same function.

See {@tutorial tags.inlineid.pkg} and {@tutorial tags.inlinetoc.pkg} for detailed information

In the example below, {@}id} is used to name the refsect1 "mysection" and the refsect2 "mysection.mysubsection" - note that the sub-sections inherit the parent section's id.

For an in-depth look at phpDocumentor tags, read {@tutorial tags.pkg}, and for an in-depth look at inline tags, read {@tutorial inlinetags.pkg}.

Documenting your PHP project

Where to begin

Before you begin documenting your PHP project, you might want to try a test run on your undocumented source code to see what kind of information phpDocumentor will automatically extract from the source code. phpDocumentor is designed to make the task of documenting minimally redundant. This means less work, and better, up-to-date documentation with less effort than it used to take.

Caution phpDocumentor is a great tool, but it will not write good documentation for you. Please read the {@tutorial phpDocumentor.pkg}

Elements of the source code that can be documented

Dividing projects into packages

To understand the role of packages and how to use {@tutorial tags.package.pkg}, it is important to know the logic behind packaging in PHP. The quest for structured programming led to the invention of functions, then classes, and finally packages. Traditionally, a re-usable software module was a collection of variables, constants and functions that could be used by another software package. PHP is an example of this model, as their are many extensions that consist of constants and functions like the tokenizer extension. One can think of the tokenizer extension as a package: it is a complete set of data, variables and functions that can be used in other programs. A more structured format of this model is of course objects, or classes. A class contains variables and functions (but no constants in PHP). A single class packages together related functions and variables to be re-used.

phpDocumentor defines package in two ways:

1. Functions, Constants and Global Variables are grouped into files (by the filesystem), which are in turn grouped into packages using the @package tag in a page-level DocBlock
2. Methods and Class Variables are grouped into classes (by php), which are in turn grouped into packages in a Class DocBlock

These two definitions of package are exclusive. In other words, it is possible to have classes of a different package of the file that contains it! Here's an example:

Caution It may be possible, but don't put classes into a different package from the file they reside in, that will be very confusing and unnecessary. This behavior is deprecated, in version 2.0, phpDocumentor will halt parsing upon this condition.

```
<?php
/**
 * Pretend this is a file
```

```
 *
 * Page-level DocBlock is here because it is the first DocBlock
 * in the file, and is immediately followed by the second
 * DocBlock before any documentable element is declared
 * (function, define, class, global variable, include)
 * @package pagepackage
 */
/**
 * Here is the class DocBlock
 *
 * The class package is different from the page package!
 * @package classpackage
 */
class myclass
{
}
?>
```

For more information on page-level versus class-level packaging, see {@tutorial elements.pkg#procedural}

Perhaps the best way to organize packages is to put all procedural code into separate files from classes. {@link http://pear.php.net PEAR} recommends putting every class into a separate file. For small, utility classes, this may not be the best solution for all cases, but it is still best to separate packages into different files for consistency.

Advanced phpDocumentor: tutorials and extended Documentation

phpDocumentor developers have received a number of requests to allow linking to external documentation, and to sections of that documentation. If phpDocumentor only could create HTML documents, this would be a simple task, but the presence of PDF and now XML converters as well as future possibilities complicates this question tremendously. What is the solution?

Give phpDocumentor the ability to parse external documentation in a common format and then convert it to the appropriate format for each converter. The implementation of this solution in version 1.2.0 is very versatile. Making use of the standard DocBook XML format, external documentation can be designed and then reformatted for any output. No longer is external documentation tied down to one "look." Here's a short list of the benefits of this approach:

Benefits of using DocBook as the format:

- DocBook is very similar to HTML at the basic level and very easy to learn.
- Each template has its own options.ini file which determines how the DocBook tags will be translated into the output language - no need to learn xslt.
- Adding in xslt support will be very easy to allow for future customization

Benefits of integrating tutorials/external documentation into phpDocumentor:

- Linking to external documentation from within API docs is possible
- Linking to API docs from external documentation is also possible
- Customizable table of contents, both of all tutorials and within a tutorial via {@tutorial tags.inlinetoc.pkg}
- It is possible to create User-level documentation that has direct access to Programmer-level documentation

User-level documentation generally consists of tutorials and information on how to use a package, and avoids extreme detail. On the other hand, programmer-level documentation has all the details a programmer needs to extend and modify a package. phpDocumentor has been assisting the creation of programmer-level documentation since its inception. With the addition of tutorials, it can now ease the creation of user-level documentation.

For an in-depth look at how to use tutorials, read {@tutorial tutorials.pkg}

Running phpDocumentor

There are two bundled ways of invoking phpDocumentor, the command-line phpdoc, or the web interface phpdoc.php/new_phpdoc.php.

Using the new Web Interface docbuilder

The new web interface requires a working installation of PHP with a web server, and must be accessible from the document root. Docbuilder is the creation of Andrew Eddies, and it combines some slick formatting with the functionality of the old web interface. The docbuilder interface can be accessed via index.html in the install directory of phpDocumentor, or the docbuilder subdirectory.

Using the old Web Interface {@link phpdoc.php} or {@link new_phpdoc.php}

In order to use the web interface, there are a number of factors that must be set up before anything else can occur. First, you need a working web server with php (it had to be said). This manual will not assist with that setup. Next, the phpdoc.php or new_phpdoc.php file needs to be accessible by the webserver. In unix, place a symbolic link using ln -s to the desired interface into the public html directory.

Caution Security is always an issue with the internet. Do not place phpDocumentor into the web server publicly available path on a server connected to the internet. Make sure that phpDocumentor will not have the power to overwrite *ANY* system or user files.

Note that since the webserver runs as user nobody in unix, the generated files will be owned by nobody. The only way to change this is to either run phpDocumentor from the command-line or to add a chuser wrapper around httpd. *We do not recommend using a chuser wrapper or running phpDocumentor as root.* It is much easier and safer to use a config file (see {@tutorial phpDocumentor.howto.pkg#using.config-files}) from the command line.

Using the Command-line tool

Running the command-line in MS Windows

Running phpDocumentor from the command-line is fairly straightforward, even in windows. It is recommended to use the web interface in windows, from a non-publicly accessible server root, as none of the permissions issues exist that exist in unix. However, to use phpDocumentor from the command line is possible. First, put the location of php into the system path, then type:

```
C:\>php-cli C:\path\to\phpdoc\phpdoc [commandline]
```

An alternative is to edit the phpdoc.bat file, and place phpdoc in the path, then you can run phpdoc as a normal command.

Running the command-line in unix

Running the command-line tool phpdoc is very easy in unix:

```
.\phpdoc [commandline]
```

-c, --config

Use this option to load a config file (see {@tutorial phpDocumentor.howto.pkg#using.config-files})

"phpdoc -c default" will load the default.ini file

Command-line switches		
	-cp	--converterparams
	-ct	--customtags
-d	--directory	name of a directory(s) to parse directory1,directory2
-dc	--defaultcategoryname	name to use for the default category. If not specified, uses 'default'
-dh	--hidden	set equal to on (-dh on) to descend into hidden directories (directories starting with '.'), default is off
-dn	--defaultpackagename	name to use for the default package. If not specified, uses 'default'
-ed	--examplesdir	full path of the directory to look for example files from @example tags
-f	--filename	name of file(s) to parse ',' file1,file2. Can contain complete path and * ? wildcards
-i	--ignore	file(s) that will be ignored, multiple separated by ','. Wildcards * and ? are ok
-it	--ignore-tags	tags to ignore for this parse. @package, @subpackage, @access and @ignore may not be ignored.
-j	--javadocdesc	use JavaDoc-compliant description (short desc is part of description, and is everything up to first .)
-o	--output	output information, format:converter:template (HTML:frames:phpedit for example)
-p	--pear	Parse a PEAR-style repository (package is directory, _members are @access private) on/off default off
-po	--packageoutput	output documentation only for selected packages. Use a comma-delimited list
-pp	--parseprivate	parse elements marked private with @access. Valid options are "on" and "offn" default value is "off"
-q	--quiet	do not display parsing/conversion messages. Useful for cron jobs
-s	--sourcecode	generate highlighted sourcecode for every parsed file (PHP 4.3.0+ only) on/off default off
-t	--target	path where to save the generated files
-ti	--title	title of generated documentation, default is 'Generated Documentation'
-tb	--templatebase	base location of all templates for this parse. Note that if -tb /path/to/here, then templates for HTML:frames:default must be in /path/to/here/Converters/HTML/frames/templates/default/templates and the /path/to/here/Converters/HTML/frames/templates/default/templates_c directory must exist, or Smarty will bail on attempting to compile the templates.

-cp, --converterparams

This option is only used to pass dynamic parameters to extended converters. The options passed should be separated by commas, and are placed in the global variable `$_phpDocumentor_setting['converterparams']`, an array. It is the responsibility of the Converter to access this variable, the code included with phpDocumentor does nothing with it.

-ct, --customtags

Use this option to specify tags that should be included in the list of valid tags for the current run of phpdoc

"phpdoc -ct mytag,anothertag" will tell phpDocumentor to parse this DocBlock:

```
/**
 * @mytag this is my tag
 * @anothertag this is another tag
 */
```

without raising any errors, and include the tags in the known tags list for the template to handle. Note that in version 1.2.0+, the unknown_tags array is passed to templates.

-dn, --defaultcategoryname

This will tell phpDocumentor to group any uncategorized elements into the primary categoryname. If not specified, the primary category is "default." The category should be specified using the {@tutorial tags.category.pkg} tag.

```
/**
 * This package has no category and will be grouped into the default category unless -dc  ↩
    categoryname is used
 * @package mypackage
 */
class nocategory
{
}
```

-dn, --defaultpackagename

Use this option to tell phpDocumentor what the primary package's name is. This will also tell phpDocumentor to group any unpackaged elements into the primary packagename. If not specified, the primary package is "default"

```
/**
 * This class has no package and will be grouped into the default package unless -dn  ↩
    packagename is used
 */
class nopackage
{
}
```

-d, --directory

This or the -f option must be present, either on the command-line or in the config file.

Unlike -f, -d does not accept wildcards. Use -d to specify full paths or relative paths to the current directory that phpDocumentor should recursively search for documentable files. phpDocumentor will search through all subdirectories of any directory in the command-line list for tutorials/ and any files that have valid .php extensions as defined in phpDocumentor.ini. For example:

"phpdoc -d relative/path/to/dir1,/fullpath/to/here,.."

-ed, --examplesdir

The -ed option is used to specify the full path to a directory that example files are located in. This command-line setting affects the output of the {@tutorial tags.example.pkg} tag.

"phpdoc -ed /fullpath/to/examples"

-f, --filename

This or the -d option must be present, either on the command-line or in the config file.

This option is used to specify individual files or expressions with wildcards * and ?. Use * to match any string, and ? to match any single character.

- phpdoc -f m*.p* will match myfile.php, mary_who.isthat.phtml, etc.
- phpdoc -f /path/m* will match /path/my/files/here.php, /path/mommy.php /path/mucho/grande/what.doc, etc.
- phpdoc -f file?.php will match file1.php, file2.php and will not match file10.php

-h, --hidden

Use this option to tell phpDocumentor to parse files that begin with a period (.)

-i, --ignore

Use the -i option to exclude files and directories from parsing. The -i option recognizes the * and ? wildcards, like -f does. In addition, it is possible to ignore a subdirectory of any directory using "dirname/"

- phpdoc -i tests/ will ignore /path/to/here/tests/* and /path/tests/*
- phpdoc -i *.inc will ignore all .inc files
- phpdoc -i *path/to/* will ignore /path/path/to/my/* as well as /path/to/*
- phpdoc -i *test* will ignore /path/tests/* and /path/here/my_test.php

-it, --ignore-tags

Use the -it option to exclude specific tags from output. This is used for creating multiple sets of documentation for different audiences. For instance, to generate documentation for end-users, it may not be desired to output all @todo tags, so --ignore-tags @todo would be used. To ignore inline tags, surround them with brackets {} like --ignore-tags {@internal}.

--ignore-tags will not ignore the core tags @global, @access, @package, @ignore, @name, @param, @return, @staticvar or @var.

Caution The --ignore-tags option requires a full tag name like --ignore-tags @todo. --ignore-tags todo will not work.

-j, --javadocdesc

Use this command-line option to enforce strict compliance with JavaDoc. JavaDoc DocBlocks are much less flexible than phpDocumentor's DocBlocks (see {@tutorial phpDocumentor.howto.pkg#basics.docblock} for information).

-o, --output

Use this required option to tell phpDocumentor which Converters and templates to use. In this release, there are several choices:

- HTML:frames:* - output is HTML with frames.
  - HTML:frames:default - JavaDoc-like template, very plain, minimal formatting
  - HTML:frames:earthli - BEAUTIFUL template written by Marco von Ballmoos
  - HTML:frames:l0l33t - Stylish template
  - HTML:frames:phpdoc.de - Similar to phpdoc.de's PHPDoc output
  - HTML:frames:phphtmllib - Very nice user-contributed template
  - all of the templates listed above are also available with javascripted expandable indexes, as HTML:frames:DOM/name where name is default, l0l33t, phpdoc.de, etcetera
  - HTML:frames:phpedit - Based on output from {@link http://www.phpedit.net PHPEdit} Help Generator
- HTML:Smarty:* - output is HTML with no frames.
  - HTML:Smarty:default - Bold template design using css to control layout
  - HTML:Smarty:HandS - Layout is based on PHP, but more refined, with logo image
  - HTML:Smarty:PHP - Layout is identical to the PHP website
- CHM:default:* - output is CHM, compiled help file format (Windows help).
  - CHM:default:default - Windows help file, based on HTML:frames:l0l33t
- PDF:default:* - output is PDF, Adobe Acrobat format
  - PDF:default:default - standard, plain PDF formatting
- XML:DocBook:* - output is XML, in DocBook format
  - XML:DocBook/peardoc2:default - documentation ready for compiling into peardoc for online pear.php.net documentation, 2nd revision

In 1.2.0+, it is possible to generate output for several converters and/or templates at once. Simply specify a comma-delimited list of output:converter:template like:

```
phpdoc -o HTML:frames:default,HTML:Smarty:PHP,HTML:frames:phpedit,PDF:default:default -t ./ ↩
    docs -d .
```

-pp, --parseprivate

By default, phpDocumentor does not create documentation for any elements marked {@tutorial tags.access.pkg} private. This allows creation of end-user API docs that filter out low-level information that will not be useful to most developers using your code. To create complete documentation of all elements, use this command-line option to turn on parsing of elements marked private with @access private.

-po, --packageoutput

Use this option to remove all elements grouped by {@tutorial tags.package.pkg} tags in a comma-delimited list from output. This option is useful for projects that are not organized into separate files for each package. Parsing speed is not affected by this option, use {@tutorial phpDocumentor.howto.pkg#using.command-line.ignore} whenever possible instead of --packageoutput.

```
phpdoc -o HTML:frames:default -t ./docs -d . -po mypackage,thatotherpackage
```

will only display documentation for elements marked with "@package mypackage" or "@package thatotherpackage"

-p, --pear

Use this option to parse a {@link http://pear.php.net PEAR}-style repository. phpDocumentor will do several "smart" things to make life easier:

- PEAR-style destructors are automatically parsed
- If no @package tag is present in a file or class DocBlock, it will guess the package based on subdirectory
- If a variable or method name begins with "_", it will be assumed to be {@tutorial tags.access.pkg} private, unless an @access tag is present. Constructors are also assumed to be @access private in the current version.

@access and @package should always be explicit!
Warnings will be raised for every case above, as @package should always be explicit, as should @access private. We do not like to assume that phpDocumentor knows better than you what to do with your code, and so will always require explicit usage or raise warnings if phpDocumentor does make any assumptions.

-q, --quiet

This option tells phpDocumentor to suppress output of parsing/conversion information to the console. Use this for cron jobs or other situations where human intervention will not be needed.

-s, --sourcecode

This option tells phpDocumentor to generate highlighted cross-referenced source code for every file parsed. The highlighting code is somewhat unstable, so use this option at your own risk.

-t, --target

Use this to tell phpDocumentor where to put generated documentation. Legal values are paths that phpDocumentor will have write access and directory creation priveleges.

-tb, --templatebase

-tb accepts a single pathname as its parameter. The default value of -tb if unspecified is <phpDocumentor install directory>/phpDocument
This raises a very important point to understand. The -tb command-line is designed to tell phpDocumentor to look for *all* templates for all converters in subdirectories of the path specified. By default, templates are located in a special subdirectory structure.

For the hypothetical outputformat:convertername:templatename {@tutorial phpDocumentor.howto.pkg#using.command-line.output} argument, the directory structure is Converters/outputformat/convertname/templates/templatename/. In addition, Smarty expects to find two subdirectories, templates/ (containing the Smarty template files) and templates_c/ (which will contain the compiled template files).

"phpdoc -tb ~/phpdoctemplates -o HTML:frames:default" will only work if the directory ~/phpdoctemplates/Converters/HTML/frames/default/templates contains all the template files, and ~/phpdoctemplates/Converters/HTML/frames/default/templates_c exists.

-ti, --title

Set the global title of the generated documentation

phpDocumentor's dynamic User-defined config files

The new {@tutorial phpDocumentor.howto.pkg#using.command-line.config} command-line options heralds a new era of ease in using phpDocumentor. What used to require:

```
phpdoc -t /path/to/output -d path/to/directory1,/another/path,/third/path -f /path/to/ ↩
    anotherfile.php -i *test.php,tests/ -pp on -ti My Title -o HTML:frames:phpedit
```

Can now be done in a single stroke!

```
phpdoc -c myconfig
```

What makes this all possible is the use of config files that contain all of the command-line information. There are two configuration files included in the distribution of phpDocumentor, and both are in the user/ subdirectory. To use a configuration file "myconfig.ini," simply place it in the user directory, and the command-line "phpdoc -c myconfig" will tell phpDocumentor to read all the command-line settings from that config file. Configuration files may also be placed in another directory, just specify the full path to the configuration file:

```
phpdoc -c /full/path/to/myconfig.ini
```

The best way to ensure your config file matches the format expected by phpDocumentor is to copy the default.ini config file, and modify it to match your needs. Lines that begin with a semi-colon (;) are ignored, and can be used for comments.

phpDocumentor.ini - global configuration options

The phpDocumentor.ini file contains several useful options, most of which will never need to be changed. However, some are useful. The section [_phpDocumentor_options] contains several configuration settings that may be changed to enhance output documentation. For Program Location, the relative path is specified as Program_Root/relativepath/to/file. The prefix "Program_Root" may be changed to any text, including none at all, or "Include ", etc. The [_phpDocumentor_setting] section can be used to specify a config file that should be used by default, so phpDocumentor may be run without any command-line arguments whatsoever! The [_phpDocumentor_phpfile_exts] section tells phpDocumentor which file extensions are php files, add any non-standard extensions, such as "class" to this section.