1 phpDocumentor Tutorials

phpDocumentor Tutorials — Writing User-level documentation/tutorials with phpDocumentor

Synopsis

{@toc}

What are phpDocumentor-style tutorials/extended documentation?

New in version 1.2.0+, tutorials are the missing link in auto-documentation. We've had many requests from developers to be able to include links to external documentation, such as the reference of template tags for a template engine. The source code would be harder to read if this kind of information was spread along the 150-300 lines it would take to describe the usage. phpDocumentor Tutorials solve the problem of including external documentation by leveraging existing tried-and-true XML DocBook format and the power of phpDocumentor's inline tags (see {@tutorial inlinetags.pkg}). Now, it is possible to cross-reference between external documentation and generated API documentation through the versatile {@tutorial tags.inlinelink.pkg}, {@tutorial tags.inlinetutorial.pkg}, and {@tutorial tags.tutorial.pkg} tag.

How to write phpDocumentor-style tutorials/extended documentation

Tutorials may be written as any legal DocBook top-level tag (<book>, <chapter>, <article>, <refentry>), but it is *highly* recommended for any project that may become a part of PEAR to write with the <refentry> tag as the top-level, as this will automatically translate into peardoc2-ready tutorials. Other projects have no such constraint and may do anything they wish.

Inline Tags that may be used in tutorials

There are several inline tags that may be used in tutorials. The {@tutorial tags.inlinelink.pkg} and {@tutorial tags.inlinetutorial.pkg} tags should be used in the same manner as in API source. Inline tags that are specific to tutorials include {@tutorial tags.inlineid.pkg} and {@tutorial tags.inlinetoc.pkg}. Inline {@}id} is used to mark areas in a similar manner to the way that <a name="section"> in HTML. {@}id} will automatically generate an id tag appropriate for the output format. The {@}toc} tag is used at the top of a tutorial to generate an internal table of contents for that tutorial. The {@}toc} tag extracts all sections that have an {@}id} tag and uses the title of that section for the generated table of contents. {@}id} is also used to mark subsections to allow linking to a sub-section of a tutorial, much the way that in HTML file.html#subsection is used.

Where to put tutorials/extended documentation so phpDocumentor will find it

To simplify linking of tutorials, all tutorials must be in a subdirectory named "tutorials" and must be passed in a {@tutorial phpDocumentor.howto.pkg#using.command-line.directory} or {@tutorial phpDocumentor.howto.pkg#using.command-line.filename} command-line switch.

In addition, tutorials are grouped by the same packages that group documented elements (see {@tutorial phpDocumentor.howto.pkg#doc for more information). To associate a tutorial with a package, place it in a subdirectory of tutorials/ named after the package. A tutorial for package "phpDocumentor" goes in "tutorials/phpDocumentor," a tutorial for subpackage "CHMDefault" of package "Converters" goes in "tutorials/Converters/CHMdefault/" - for an example, see the tutorials/ subdirectory of phpDocumentor.

Naming tutorials: .pkg, .cls, .proc

There are three ways of associating tutorials/extended documentation with the elements they are written for:

- Package-level tutorials always have file extension "pkg"
- Class-level tutorials always have file extension "cls"
- Procedural-level tutorials always have file extension "proc"

Package-level documentation

The primary documentation for a package or subpackage should go in a file with the same name as the package or subpackage. For package "phpDocumentor" the primary package-level documentation is in "tutorials/phpDocumentor/phpDocumentor.pkg" For subpackage "CHMdefault" of package "Converters," the primary subpackage-level documentation is in "tutorials/Converters/CHMDefault/CHMdefault.pkg"

Other package-level documentation can have any filename, but must have the package-tutorial extension .pkg.

Class-level documentation

Similar to package-level documentation, the primary documentation for a class must have the same name as the class. The primary documentation for the class "Converter" in package "Converters" will be found in "tutorials/Converters/Converter.cls."

Procedural-level documentation

If a package contains a number of functions, it may be good to include procedural-level documentation. If a particular file contains a large amount of procedural information that must be documented, it is possible to link tutorials to that file's API docs in the same manner it is possible to link to a class's API docs.

The primary documentation for the procedural component of a PHP file must have the same name as the file. The primary procedural documentation for the file test.php in package "examples" will be found in "tutorials/examples/test.php.proc."

Creating a tutorial hierarchy

If you've ever wanted to separate a large user-level document into separate documents the way the PHP manual does, you're in luck. phpDocumentor provides a dynamic and flexible solution to this problem. When parsing a tutorial, phpDocumentor looks for an .ini file associated with that tutorial in the same directory. In other words, for "tutorials/phpDocumentor/phpDocumentor.pkg," phpDocumentor will search for a file named "tutorials/phpDocumentor/phpDocumentor.pkg.ini" and will parse out a section named [Linked Tutorials]. This section defines every child tutorial of the tutorial phpDocumentor.pkg

In order for a tutorial to be linked to another tutorial as a child, it must be in the same package, subpackage and be the same tutorial type. tutorials/phpDocumentor/phpDocumentor.pkg cannot have tutorials/phpDocumentor/oops.cls as a child tutorial, nor can it have tutorials/phpDocumentor/oops/almostworks.pkg as a child tutorial. It can have tutorials/phpDocumentor/works.pkg as a child, because the tutorial is in the same package, and is package-level documentation.

To help enforce this rule, the [Linked Tutorials] section is a list of filenames minus path and extension.

Here's an example tutorials/test/test.pkg.ini:

---

phpDocumentor will search for tutorials/test/child1.pkg, tutorials/test/child2.pkg, and tutorials/test/child3.pkg and link them in that order. Note that any tutorial can have an .ini file, allowing unlimited depth of tutorials. We don't recommend going beyond 3 sub-levels, that will probably confuse readers more than it helps them.

For a working example, check out the .ini files in phpDocumentor's own tutorials/ directory.

Converting tutorials/extended documentation to HTML/CHM/PDF/XML/...

Support is built into the existing Converters for automatic conversion of tutorials to the desired output without any extra work on your part. However, if the output is not quite right, and something needs to be changed, this is very easy. For every template of every converter, there is a file in the template directory called "options.ini." In other words, for the HTML:frames:default template, the options.ini file is phpDocumentor/Converters/HTML/frames/templates/default/options.ini. Open this file to see all of the template options for conversion of data.

The first section of the .ini file, [desctranslate], is for conversion of DocBlocks and is covered elsewhere (see {@tutorial phpDocumentor.howto.pkg#basics.desc}).

After that comes the section that deals with tutorials, [ppage].

[ppage] section of options.ini

The [ppage] section contains simple rules for transforming DocBook tags and DocBook entities into any output. Re-ordering is supported for attributes and titles only. The best way to learn how to use this simple and powerful transformation is to study the options.ini files for HTML:frames:phpedit and PDF:default:default

DocBook Entity translation

The translation of entities like ” is handled through entries in the [ppage] section like the following for html:

```
[ppage]
&quot; = &quot;
&amp; = &amp;
&rdquo; = &rdquo;
```

or the following, for PDF:

```
[ppage]
&quot; = """
&amp; = &
&rdquo; = """
```

Note that to specify a ", it must be enclosed in double quotes, hence the usage of """

DocBook Tag translation

Each tag is transformed using a few possibilities. If a tag is to be simply transformed into another tag (as in to <para></para>, use

- tagname = newtagname

phpDocumentor will automatically enclose the tag contents with the new tagname and all of its attributes

If the start and endtag should be different, specify the exact text using a slash before the tagname as in:

- tagname = <starttext attr="myval" />
- /tagname = "\n"

If a tag needs only a new tag and attribute name (for example link and linkend become a and href):

- tagname = newtagname
- tagname->attr = newattrname

If a tag needs only a new tag and attribute name, and a new attribute value (for example table frame="all" becomes table frame="border"):

- tagname = newtagname
- tagname->attr = newattrname
- tagname->attr+val = newval

If a tag needs only a new tag and attribute name, and two attributes combine into one (for example table colsep="1" rowsep="1" becomes table rules="all"):

- tagname = newtagname
- tagname->attr = newattrname
- tagname->attr2 = newattrname
- tagname->attr+val|attr2+val = newval

If an attribute should be ignored:

- tagname = newtagname
- tagname!attr =

If all attributes should be ignored:

- tagname = newtagname
- tagname! =

If a tag should have all attributes and be a single tag (like
:

- tagname = newtagname
- tagname/ =

If an attribute should be changed to a new name for all cases (like role changed to class, which would be $attr$role = class):

- $attr$attrname = newattrname

Tips for using the .ini files

The parsing of these ini files is performed by {@link phpDocumentor_parse_ini_file()}, a much more powerful version of PHP's {@link parse_ini_file()} built-in function. If a value is encased in quotation marks, it will be stripped of backslashes (\) and used as is. In other words:

---

will parse out as tagname = Some "enclosed" text.

Otherwise, this example:

---

will parse out as tagname = Some "enclosed"\n\ttext.

Note that escaping of quotation marks is not needed, phpDocumentor_parse_ini() only strips the outer tags and then performs {@link stripcslashes()} on the remaining string.

What next?

Write your tutorials and extended documentation!