How to Contribute to the Documentation

How to contribute to the mod_perl documentation: style, tools, etc.

Last modified Thu Jan 29 08:43:30 2004 GMT

Table of Contents:

1 Documentation Style Guide

1.1 Description

This document defines the style the authors should follow when writing a documentation for the mod_perl documentation project.

1.2 Formatting

The documentation format is plain POD (Plain Old Documentation), which then will be converted into HTML, PS, PDF and other formats. You can learn the syntax of this format from the *perlpod* manpage and the new *perlpodspec* manpage from 5.8 versions of Perl.

1.3 Document structure

The document should be constructed from at least the following `=head1` sections.

- **NAME**

  The first section's title must be `NAME` with a short title as a content. e.g.:

  ```
  =head1 NAME

  This is the title of the document
  ```

  There should be no POD escape characters in this section, since it can be used in places where it's not possible to render things like I<> or B<>.

  This section won't appear in the finally rendered document, except as the title of the document.

- **DESCRIPTION**

  `DESCRIPTION` or `Description`, so it gets rendered like other =head sections in the document in case you don't use upper case for these.

  The first paragraph of this section will be used on the index pages that link the documents together. e.g.:

  ```
  =head1 Description

  This document explains...
  ```

  This section must appear in the first three sections of the document. It's not required to be the one following the `NAME` section since in Perl modules pods it usually comes third after the `SYNOPSIS` section.

- **Author**

  The author of the document with an optional email address or other means for reaching the author.

Usually comes as one of the last sections of the document.

1.4 Conventions

Please try to use the following conventions when writing the docs:

- When using domain names in examples use only *example.com* and its derivatives (e.g. *foo.example.com*) or *localhost* (or *localhost.localdomain*). *example.com* is an official example domain.

- Keep the text width <= 74 cols.

- Please avoid leaving ^M (CR on the DOS platforms). Either use the editor to remove these new line chars, or use Perl:

  ```
  % perl -pi -e 's|\cM||' filename.pod
  ```

  If you want to iterate over all files in a directory:

  ```
  % find . -type f -exec perl -pi -e 's|\cM||' {} \;
  ```

  though watch for binaries, like images or the *cache.*.dat* files left by DocSet, which may get corrupted with the above command. So something like this more fine tuned command is safer:

  ```
  % find . -type f -name "*.pod" -exec perl -pi -e 's|\cM||' {} \;
  ```

- Use `C<Module>` for module names, directives, function names, etc. If correcting older documentation, remember not to leave any quotes around the old names (for example, don't do C<"GET">, but just C<GET>).

  Some older documentation uses B<> for module names. This was because `pod2man` didn't make C<> stand out enough. If you spot any of these, please replace them with C<>. Use B<> for stressing very important things. Use them infrequently, since if there are many phrases in bold the original intention is getting lost.

- Use I<filename> for filenames, URIs and things that are generally written in italics.

- Use B<stress> for stressing things, but you should avoid using this tag unless you think things are very important. Over-use of the bold text reduces it's original intention -- make things stand out.

- Use E<gt> for encoding `$r->filename` as in C<$r-E<gt>filename>. Note that with some Perl versions `pod2html(1)` and some other `pod2*` are broken and don't correctly interpret this tag.

- URLs are left unmarked. Pod2Html automatically identifies and highlights them. If later we would like to do that inline, we can have an easy `s///` one liner.

- Linking between items in the same doc and across documents: Currently use the technique explained in perlpod man page. It's not very sophisticated, but we will have to think about some better technique.

Currently, you can do this: for example, if editing the document *guide/performance.pod*, you can link to the *install.pod* one by using

```
L<installation instructions|guide::install>
```

or

```
L<installation instructions|docs::1.0::guide::install>
```

You may also link to the index of a section by using

```
L<The Guide|guide::index>
```

As you can see in the base *config.cfg* file, there are some search paths used to make linking more comfortable. That is why you can, for some documents, use absolute links (à la `docs::1.0::guide::install`) and relative links (`guide::install`).

- Command line examples. Please use the following prompts to be consistent.

  user mode prompt:

  ```
  % ls -l
  ```

  root mode prompt:

  ```
  # ls -l
  ```

  This is also documented in the Conventions document. If there is possible confusion about whether the second one is a root prompt or a comment, it might be a good idea to indicate it.

  For Operating System-specific information, use an adapted prompt: for example for Win32:

  ```
  C:\> bin\build
  ```

- Titles and subtitles:

  Use the head tags:

  ```
  =head(1,2,3...)
  ```

  Please try to avoid titles in **ALLCAPS**. Use caps like **This**, which are a little more *normal*. If porting old documents, correct this.

- Code examples:

  META: not implemented yet! Currently use F<>

  A new pod tag:

```
=example 1.1 This is a title
```

becomes:

```
<p><i>Example 1.1: This is a title</i></p>
```

- Figures (images):

  META: not implemented yet! Currently use =for html

  A new pod tag:

```
=figure figure1.1.png This is a title
```

  becomes:

```
<p><center><img src="figure1.1.png"></center></p>
<p><center><b>Figure 1.1: This is a title</b></center></p>
```

  The index is extracted automatically from the file name.

- META: not implemented yet!

  Footnotes. These aren't defined in the current perlpod yet. So please use [F] footnote [/F] semantics and later we will come up with some way to make it a correct footnote.

- META: not implemented yet!

  Sidebars. Just like footnotes - it's not defined yet. Use [S] sidebar [/S] for now.

- Paragraphs.

  Try to keep the paragraphs not too long as it is hard to read them if they are too long. Follow common sense for that.

  Paragraphs are separated by an empty new line. Please make sure that you don't leave any spaces on this line. Otherwise the two paragraphs will be rendered as one. Especially remember to put a new empty line between text and code snippets.

- **Code snippets**

  As you know in POD if you want something to be left untouched by the translator, you have to insert at least one space before each line. Please use the 2 space indent to specify the text snippet and for the code examples please use the 4 spaces indentation. No tabs please.

  Also remember that if the code snippet includes empty lines, you should prefix these with 2 spaces as well, so the examples will be continuous after they get converted into other formats.

Here is an example:

```
my $foo;
for (1..10) {
     $foo += $_;

     if ($foo > 6) {
         print "foo\n";
     }
     else {
         print "bar\n";
     }
}
```

From this example you can learn other style details that you should follow when writing docs. In any case, follow the mod_perl coding guidelines for code.

- **Automatic code extraction**

  The documentation includes numerous code snippets and complete scripts, you don't want the reader to type them in, so we want to make all the code available to the reader in a separated files, located in each chapter's parent's directory (e.g. ch2/ex2.pl)

  Well at the beginning you might think that it might be a good idea to keep all the code in sync with the doc, but very soon you will find out that you change the code in the text and move the chapters and sections around, which makes it impossible to maintain the external source code.

  So what we have to do (and I haven't made it yet) is to use a convention for the code to be automatically extracted, e.g.:

```
file:example.pl
---------------
#!/usr/bin/perl -w

use CGI;
my $q = new CGI;

print "Hi";
```

  So as I've said before we must not forget to add 2 space characters indentation to empty lines with no code in them, so that the parser picks up the whole code, removes the header with the filename and separator, puts back the code itself, saves it to the filename written at the top, and places it into the same directory name the text is located in. (Well it can be a separate tree for the code). If there are real empty lines, only part of the script will be saved, which will make the release broken. Another approach is to add some tail (ending token), but it's a mess I think. I develop the text using *cperl-mode.el* in xemacs which shows all space characters not followed by any text - this helps a lot!

- **Documenting Important Changes**

  If you are posting a patch or a committing a patch, please document the important changes that would be of interest to the end user. For more info please read the Changes file doc.

- **Naming mod_perl major versions**

  We have adopted the convention that mod_perl major versions should be named as *1.0* and *2.0*, and not *1.x* and *2.x*.

1.5 Review process

If you want to send a review of a document to the document maintainer, please follow the following guidelines:

1.5.1 Diff or not Diff?

Since the text is changing a lot it's much harder to apply patches, especially if you happen to make a patch against an older version.

Therefore we have found that it's much better for the docs maintainers to receive the whole document which is already corrected the way you think it should be and possible extra comments, as explained in the next section.

Once we receive such a document we can use visual diff programs to make an easy merge, even if the document that you have modified has been changed since then. I suggest to use emacs's `Ediff` module for visual merges. I'm sure other editors provide similar functionality.

[Stas: if you know about these functionalities, please let me know so we can share the knowledge with others who don't use emacs.]

To submit normal patches (when they are minor changes, and you're sure the document hasn't changed), use the `cvs diff` method:

```
% cvs diff -u src/docs/1.0/...pod
```

If you're adding a file, especially if it needs a new directory, it might be a good idea to submit a patch against `/dev/null`, which will automatically create the new directory, like this.

```
% diff -u /dev/null newdir/newfilename.pod
```

Or on Windows:

```
% diff -u NUL newdir/newfilename.pod
```

1.5.2 Adding Inline Remarks

- TODO semantics:

  I've gotten used to (META: do something) approach since the old days when I read the linux documentations. So you will see lots of META tags scattered around the sources. It makes it easy to see what things aren't yet complete and mark things that we want to work on later. So please use something like:

```
[META: this should be completed]
```

- Review Comments:

  If you review some document and you want to comment on something, just embed a paragraph with your comments enclosed in [] and with your name prepended. E.g:

  ```
  [Stas: This document needs a review.
  But it looks OK after all.]
  ```

  if your first name is a common one, please use the last name as well, or some other way to easily identify you so the maintainer of the document can contact you for an additional questions.

1.6 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

Stas Bekman <stas (at) stason.org>

1.7 Authors

- **Stas Bekman <stas (at) stason.org>**

2 Changes File Specs

2.1 Description

This doc clears the confusion regarding the need and the maintenance guidelines of *Changes.pod* files in the project.

2.2 Who Has Contributed What And When

All the modifications of every single file can be viewed via `cvs log` command. e.g., to check the history of this very file, one would run:

```
% cvs log src/contribute/docs/changes_file.pod
```

Which will display all the commit logs, who has committed the change, who has submitted the changes, etc.

2.3 The Art of Changes File

The *Changes.pod* document is not the same as the history of all changes. This document is for end user consumption, who is interested to know what are the major changes since the last time she read the documents. Or minor but important changes, like bug fixes.

Therefore the *Changes.pod* document is only needed when some sub-project goes through changes which will be of interest to the reader. Don't just add *Changes.pod* everywhere, until you really think it's needed.

The format of this document should be as dense as possible, so the reader can read through it fast and pin-point if there is something interesting to it.

There is no need to log the date every time the change is done ('cvs log' has all the info). Though it's nice to group the changes by certain milestones, so let's say every few month a time stamp is added in front of the group of the changes since the last timestamp and new changes will go to the new group. The change entries in the *docs/1.0/guide/Changes.pod* is a good example of that. In addition it used to add a version number for each milestone, which is very optional now.

This file should have the latest changes on the top.

2.4 The Scope of Changes.pod

Usually we have a separate *Changes.pod* file for each sub-set of the documents. If you feel that the changes for a few sub-sets nested in the same super-set of docs can be maintained in one file, have only one *Changes.pod*. Later if this file becomes too overloaded and its added value is getting diminished, split it into a few *Changes.pod* files placed in each sub-set. Or if you think that this will happen in the near future do this from the beginning to avoid the slicing work later.

2.5 Adding Credits

If you are the maintainer of the document, you don't have to credit each change done by you, with your name, simply leave the change entry un-credited, which automatically implies that you did that.

If someone commits something to the document maintained by someone else simply mark it with your name e.g. [Thomas Klausner]. Those who commit all the time, should pick some short (nick?)name that will distinguish them from others and make their changes with it. e.g [thomas]. The idea is to have the changes file with as little noise as possible.

There is a special case where we want to credit people who contributed very minor fixes, which don't deserve a separate changes entry. In this case just have a special entry like `Minor fixes`, where you simply list the names of those who contributed because we want to give credits to everybody. Again the *docs/1.0/guide/Changes.pod* file perfectly demonstrates that.

2.6 Sample Changes.pod

See *docs/1.0/guide/Changes.pod* as a good example.

A typical entry looks like this:

```
=head1 ???

* books: Fixed some things and then other things, and then some other
  things bla bla bla. [John Doe E<lt>john.doe (at) aol.comE<gt>]

* file: Added some content. [stas]

* otherfile: updated the "Maintenance" section, adding references to
  bla bla bla [other person]

=head1 Sat Nov 12 22:05:23 CET 2002

* docs::index : moved tutorials to "Other documentation" [stas]

* performance: minor corrections [thomas]
```

Please try to keep things correctly aligned here (ie. the first characters on each line should be vertically aligned with eachother), as this file will most often be viewed as text.

As you can see, we try to collect a number of changes, then timestamp the document like a "version".

You can use the *Changes_template.pod* as a starting point.

3 Site Maintenance

3.1 Description

This document explains how to keep the site clean.

3.2 Validation Tasks

We start from a site which is absolutely clean. Please keep it that way.

- Validate internal and external links. For example use: the *checklink.pl* from (http://validator.w3.org/checklink) I usually run the check as:

  ```
  % checklink.pl --summary --recursive --broken --quiet \
    --html -D 10 http://localhost/modperl-site > report.html
  ```

  Internal links validation also applies to POD documents. It's easy to do this, just rebuild the site with the `-l` argument to `bin/build`:

  ```
  % bin/build -lf
  ```

- Validate the correctness of the documents. The broken HTML can come from the broken source HTML document or bad templates. One of the tools that can be used is sgmlcheck. e.g.:

  ```
  % sgmlcheck dst_html/index.html
  ```

  META: anyone knows a better tool that can recursively check the whole site (ala checklink.pl) and generate an nice report?

Table of Contents: