

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 $\backslash M$ (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></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!**

Sidebar. 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 `cvcs 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>

Table of Contents:

1	Documentation Style Guide	1
1.1	Description	2
1.2	Formatting	2
1.3	Document structure	2
1.4	Conventions	3
1.5	Review process	7
1.5.1	Diff or not Diff?	7
1.5.2	Adding Inline Remarks	7
1.6	Maintainers	8
1.7	Authors	8