NAME

Pod::Simple - framework for parsing Pod

SYNOPSIS

```
TODO
```

DESCRIPTION

Pod::Simple is a Perl library for parsing text in the Pod ("plain old documentation") markup language that is typically used for writing documentation for Perl and for Perl modules. The Pod format is explained in the *perlpod* man page; the most common formatter is called "perldoc".

Pod formatters can use Pod::Simple to parse Pod documents into produce renderings of them in plain ASCII, in HTML, or in any number of other formats. Typically, such formatters will be subclasses of Pod::Simple, and so they will inherit its methods, like `parse_file`.

If you're reading this document just because you have a Pod-processing subclass that you want to use, this document (plus the documentation for the subclass) is probably all you'll need to read.

If you're reading this document because you want to write a formatter subclass, continue reading this document, and then read *Pod::Simple::Subclassing*, and then possibly even read *perlpodspec* (some of which is for parser-writers, but much of which is notes to formatter-writers).

MAIN METHODS

`$parser = SomeClass->new();`

This returns a new parser object, where *SomeClass* is a subclass of Pod::Simple.

`$parser->output_fh( *OUT );`

This sets the filehandle that `$parser`'s output will be written to. You can pass `*STDOUT`, otherwise you should probably do something like this:

```
my $outfile = "output.txt";
open TXTOUT, ">$outfile" or die "Can't write to $outfile: $!";
$parser->output_fh(*TXTOUT);
```

...before you call one of the `$parser->parse_`*whatever* methods.

`$parser->output_string( \$somestring );`

This sets the string that `$parser`'s output will be sent to, instead of any filehandle.

`$parser->parse_file(` *$some_filename* `);`

`$parser->parse_file( *INPUT_FH );`

This reads the Pod content of the file (or filehandle) that you specify, and processes it with that `$parser` object, according to however `$parser`'s class works, and according to whatever parser options you have set up for this `$parser` object.

`$parser->parse_string_document(` *$all_content* `);`

This works just like `parse_file` except that it reads the Pod content not from a file, but from a string that you have already in memory.

`$parser->parse_lines(` *...@lines...*`, undef );`

This processes the lines in `@lines` (where each list item must be a defined value, and must contain exactly one line of content -- so no items like `"foo\nbar"` are allowed). The final `undef` is used to indicate the end of document being parsed.

The other `parser_`*whatever* methods are meant to be called only once per `$parser` object; but `parse_lines` can be called as many times per `$parser` object as you want, as long as the last call (and only the last call) ends with an `undef` value.

`$parser->content_seen`

This returns true only if there has been any real content seen for this document.

SomeClass`->filter(` *$filename* `);`

SomeClass`->filter(` **INPUT_FH* `);`

SomeClass`->filter(` *\$document_content* `);`

This is a shortcut method for creating a new parser object, setting the output handle to STDOUT, and then processing the specified file (or filehandle, or in-memory document). This is handy for one-liners like this:

```
  perl -MPod::Simple::Text -e
"Pod::Simple::Text->filter('thingy.pod')"
```

SECONDARY METHODS

Some of these methods might be of interest to general users, as well as of interest to formatter-writers.

Note that the general pattern here is that the accessor-methods read the attribute's value with `$value = $parser->`*`attribute`* and set the attribute's value with `$parser->`*`attribute`*`(` *`newvalue`*`)`. For each accessor, I typically only mention one syntax or another, based on which I think you are actually most likely to use.

`$parser->no_whining(` *SOMEVALUE* `)`

If you set this attribute to a true value, you will suppress the parser's complaints about irregularities in the Pod coding. By default, this attribute's value is false, meaning that irregularities will be reported.

Note that turning this attribute to true won't suppress one or two kinds of complaints about rarely occurring unrecoverable errors.

`$parser->no_errata_section(` *SOMEVALUE* `)`

If you set this attribute to a true value, you will stop the parser from generating a "POD ERRORS" section at the end of the document. By default, this attribute's value is false, meaning that an errata section will be generated, as necessary.

`$parser->complain_stderr(` *SOMEVALUE* `)`

If you set this attribute to a true value, it will send reports of parsing errors to STDERR. By default, this attribute's value is false, meaning that no output is sent to STDERR.

Note that errors can be noted in an errata section, or sent to STDERR, or both, or neither. So don't think that turning on `complain_stderr` will turn off `no_errata_section` or vice versa -- these are independent attributes.

`$parser->source_filename`

This returns the filename that this parser object was set to read from.

`$parser->doc_has_started`

This returns true if `$parser` has read from a source, and has seen Pod content in it.

`$parser->source_dead`

This returns true if `$parser` has read from a source, and come to the end of that source.

CAVEATS

This is just a beta release -- there are a good number of things still left to do. Notably, support for EBCDIC platforms is still half-done, an untested.

SEE ALSO

Pod::Simple::Subclassing

perlpod

perlpodspec

Pod::Escapes

perldoc

COPYRIGHT AND DISCLAIMERS

AUTHOR

Original author: Sean M. Burke `sburke@cpan.org`

Maintained by: Allison Randal `allison@perl.org`