NAME

Pod::Simple::XHTML -- format Pod as validating XHTML

SYNOPSIS

```
use Pod::Simple::XHTML;

my $parser = Pod::Simple::XHTML->new();

...

$parser->parse_file('path/to/file.pod');
```

DESCRIPTION

This class is a formatter that takes Pod and renders it as XHTML validating HTML.

This is a subclass of *Pod::Simple::Methody* and inherits all its methods. The implementation is entirely different than *Pod::Simple::HTML*, but it largely preserves the same interface.

Minimal code

```
use Pod::Simple::XHTML;
my $psx = Pod::Simple::XHTML->new;
$psx->output_string(\my $html);
$psx->parse_file('path/to/Module/Name.pm');
open my $out, '>', 'out.html' or die "Cannot open 'out.html': $!\n";
print $out $html;
```

You can also control the character encoding and entities. For example, if you're sure that the POD is properly encoded (using the `=encoding` command), you can prevent high-bit characters from being encoded as HTML entities and declare the output character set as UTF-8 before parsing, like so:

```
$psx->html_charset('UTF-8');
$psx->html_encode_chars(q{&<>'"});
```

METHODS

Pod::Simple::XHTML offers a number of methods that modify the format of the HTML output. Call these after creating the parser object, but before the call to `parse_file`:

```
my $parser = Pod::PseudoPod::HTML->new();
$parser->set_optional_param("value");
$parser->parse_file($file);
```

perldoc_url_prefix

In turning *Foo::Bar* into http://whatever/Foo%3a%3aBar, what to put before the "Foo%3a%3aBar". The default value is "http://search.cpan.org/perldoc?".

perldoc_url_postfix

What to put after "Foo%3a%3aBar" in the URL. This option is not set by default.

man_url_prefix

In turning *`crontab(5)`* into http://whatever/man/1/crontab, what to put before the "1/crontab". The default value is "http://man.he.net/man".

man_url_postfix

What to put after "1/crontab" in the URL. This option is not set by default.

title_prefix, title_postfix

What to put before and after the title in the head. The values should already be &-escaped.

html_css

```
$parser->html_css('path/to/style.css');
```

The URL or relative path of a CSS file to include. This option is not set by default.

html_javascript

The URL or relative path of a JavaScript file to pull in. This option is not set by default.

html_doctype

A document type tag for the file. This option is not set by default.

html_charset

The character set to declare in the Content-Type meta tag created by default for `html_header_tags`. Note that this option will be ignored if the value of `html_header_tags` is changed. Defaults to "ISO-8859-1".

html_header_tags

Additional arbitrary HTML tags for the header of the document. The default value is just a content type header tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

Add additional meta tags here, or blocks of inline CSS or JavaScript (wrapped in the appropriate tags).

html_encode_chars

A string containing all characters that should be encoded as HTML entities, specified using the regular expression character class syntax (what you find within brackets in regular expressions). This value will be passed as the second argument to the `encode_entities` function of *HTML::Entities*. If *HTML::Entities* is not installed, then any characters other than &<"> will be encoded numerically.

html_h_level

This is the level of HTML "Hn" element to which a Pod "head1" corresponds. For example, if `html_h_level` is set to 2, a head1 will produce an H2, a head2 will produce an H3, and so on.

default_title

Set a default title for the page if no title can be determined from the content. The value of this string should already be &-escaped.

force_title

Force a title for the page (don't try to determine it from the content). The value of this string should already be &-escaped.

html_header, html_footer

Set the HTML output at the beginning and end of each file. The default header includes a title, a doctype tag (if `html_doctype` is set), a content tag (customized by `html_header_tags`), a tag for a CSS file (if `html_css` is set), and a tag for a Javascript file (if `html_javascript` is set). The default footer simply closes the `html` and `body` tags.

The options listed above customize parts of the default header, but setting `html_header` or

`html_footer` completely overrides the built-in header or footer. These may be useful if you want to use template tags instead of literal HTML headers and footers or are integrating converted POD pages in a larger website.

If you want no headers or footers output in the HTML, set these options to the empty string.

index

Whether to add a table-of-contents at the top of each page (called an index for the sake of tradition).

anchor_items

Whether to anchor every definition `=item` directive. This needs to be enabled if you want to be able to link to specific `=item` directives, which are output as `<dt>` elements. Disabled by default.

backlink

Whether to turn every =head1 directive into a link pointing to the top of the page (specifically, the opening body tag).

SUBCLASSING

If the standard options aren't enough, you may want to subclass Pod::Simple::XHMTL. These are the most likely candidates for methods you'll want to override when subclassing.

handle_text

This method handles the body of text within any element: it's the body of a paragraph, or everything between a "=begin" tag and the corresponding "=end" tag, or the text within an L entity, etc. You would want to override this if you are adding a custom element type that does more than just display formatted text. Perhaps adding a way to generate HTML tables from an extended version of POD.

So, let's say you want to add a custom element called 'foo'. In your subclass's `new` method, after calling `SUPER::new` you'd call:

```
$new->accept_targets_as_text( 'foo' );
```

Then override the `start_for` method in the subclass to check for when "$flags->{'target'}" is equal to 'foo' and set a flag that marks that you're in a foo block (maybe "$self->{'in_foo'} = 1"). Then override the `handle_text` method to check for the flag, and pass $text to your custom subroutine to construct the HTML output for 'foo' elements, something like:

```
sub handle_text {
    my ($self, $text) = @_;
    if ($self->{'in_foo'}) {
        $self->{'scratch'} .= build_foo_html($text);
        return;
    }
    $self->SUPER::handle_text($text);
}
```

handle_code

This method handles the body of text that is marked up to be code. You might for instance override this to plug in a syntax highlighter. The base implementation just escapes the text.

The callback methods `start_code` and `end_code` emits the `code` tags before and after `handle_code` is invoked, so you might want to override these together with `handle_code` if this wrapping isn't suitable.

Note that the code might be broken into multiple segments if there are nested formatting codes inside a `C<...>` sequence. In between the calls to `handle_code` other markup tags might have been emitted in that case. The same is true for verbatim sections if the `codes_in_verbatim` option is

turned on.

accept_targets_as_html

This method behaves like `accept_targets_as_text`, but also marks the region as one whose content should be emitted literally, without HTML entity escaping or wrapping in a `div` element.

resolve_pod_page_link

```
my $url = $pod->resolve_pod_page_link('Net::Ping', 'INSTALL');
my $url = $pod->resolve_pod_page_link('perlpodspec');
my $url = $pod->resolve_pod_page_link(undef, 'SYNOPSIS');
```

Resolves a POD link target (typically a module or POD file name) and section name to a URL. The resulting link will be returned for the above examples as:

```
http://search.cpan.org/perldoc?Net::Ping#INSTALL
http://search.cpan.org/perldoc?perlpodspec
#SYNOPSIS
```

Note that when there is only a section argument the URL will simply be a link to a section in the current document.

resolve_man_page_link

```
my $url = $pod->resolve_man_page_link('crontab(5)', 'EXAMPLE CRON FILE');
my $url = $pod->resolve_man_page_link('crontab');
```

Resolves a man page link target and numeric section to a URL. The resulting link will be returned for the above examples as:

```
http://man.he.net/man5/crontab
http://man.he.net/man1/crontab
```

Note that the first argument is required. The section number will be parsed from it, and if it's missing will default to 1. The second argument is currently ignored, as *man.he.net* does not currently include linkable IDs or anchor names in its pages. Subclass to link to a different man page HTTP server.

idify

```
my $id   = $pod->idify($text);
my $hash = $pod->idify($text, 1);
```

This method turns an arbitrary string into a valid XHTML ID attribute value. The rules enforced, following *http://webdesign.about.com/od/htmltags/a/aa031707.htm*, are:

- The id must start with a letter (a-z or A-Z)
- All subsequent characters can be letters, numbers (0-9), hyphens (-), underscores (_), colons (:), and periods (.).
- The final character can't be a hyphen, colon, or period. URLs ending with these characters, while allowed by XHTML, can be awkward to extract from plain text.
- Each id must be unique within the document.

In addition, the returned value will be unique within the context of the Pod::Simple::XHTML object unless a second argument is passed a true value. ID attributes should always be unique within a single XHTML document, but pass the true value if you are creating not an ID but a URL hash to point to an ID (i.e., if you need to put the "#foo" in `<a href="#foo">foo</a>`.

batch_mode_page_object_init

```
  $pod->batch_mode_page_object_init($batchconvobj, $module, $infile,
$outfile, $depth);
```

Called by *Pod::Simple::HTMLBatch* so that the class has a chance to initialize the converter. Internally it sets the `batch_mode` property to true and sets `batch_mode_current_level()`, but Pod::Simple::XHTML does not currently use those features. Subclasses might, though.

SEE ALSO

Pod::Simple, *Pod::Simple::Text*, *Pod::Spell*

SUPPORT

Questions or discussion about POD and Pod::Simple should be sent to the pod-people@perl.org mail list. Send an empty email to pod-people-subscribe@perl.org to subscribe.

This module is managed in an open GitHub repository, *https://github.com/perl-pod/pod-simple/*. Feel free to fork and contribute, or to clone *git://github.com/perl-pod/pod-simple.git* and send patches!

Patches against Pod::Simple are welcome. Please send bug reports to <bug-pod-simple@rt.cpan.org>.

COPYRIGHT AND DISCLAIMERS

ACKNOWLEDGEMENTS

Thanks to *Hurricane Electric* for permission to use its *Linux man pages online* site for man page links.

Thanks to *search.cpan.org* for permission to use the site for Perl module links.

AUTHOR

Pod::Simpele::XHTML was created by Allison Randal <allison@perl.org>.

Pod::Simple was created by Sean M. Burke <sburke@cpan.org>. But don't bother him, he's retired.

Pod::Simple is maintained by:

* Allison Randal `allison@perl.org`
* Hans Dieter Pearcey `hdp@cpan.org`
* David E. Wheeler `dwheeler@cpan.org`