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.

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.

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_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

default_title (alias: 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

TODO -- Not implemented.

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

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 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);
    } else {
        $self->{'scratch'} .= $text;
    }
}
```

SEE ALSO

Pod::Simple, *Pod::Simple::Methody*

COPYRIGHT

AUTHOR

Allison Randal <allison@perl.org>