

NAME

Pod::Simple::PullParser -- a pull-parser interface to parsing Pod

SYNOPSIS

```
my $parser = SomePodProcessor->new;
$parser->set_source( "whatever.pod" );
$parser->run;
```

Or:

```
my $parser = SomePodProcessor->new;
$parser->set_source( $some_filehandle_object );
$parser->run;
```

Or:

```
my $parser = SomePodProcessor->new;
$parser->set_source( \ $document_source );
$parser->run;
```

Or:

```
my $parser = SomePodProcessor->new;
$parser->set_source( \@document_lines );
$parser->run;
```

And elsewhere:

```
require 5;
package SomePodProcessor;
use strict;
use base qw(Pod::Simple::PullParser);

sub run {
    my $self = shift;
    Token:
    while(my $token = $self->get_token) {
        ...process each token...
    }
}
```

DESCRIPTION

This class is for using Pod::Simple to build a Pod processor -- but one that uses an interface based on a stream of token objects, instead of based on events.

This is a subclass of *Pod::Simple* and inherits all its methods.

A subclass of Pod::Simple::PullParser should define a `run` method that calls `$token = $parser->get_token` to pull tokens.

See the source for Pod::Simple::RTF for an example of a formatter that uses Pod::Simple::PullParser.

METHODS

```
my $token = $parser->get_token
```

This returns the next token object (which will be of a subclass of

Pod::Simple::PullParserToken), or undef if the parser-stream has hit the end of the document.

```
$parser->unget_token( $token )
$parser->unget_token( $token1, $token2, ... )
```

This restores the token object(s) to the front of the parser stream.

The source has to be set before you can parse anything. The lowest-level way is to call `set_source`:

```
$parser->set_source( $filename )
$parser->set_source( $filehandle_object )
$parser->set_source( \$document_source )
$parser->set_source( \@document_lines )
```

Or you can call these methods, which `Pod::Simple::PullParser` has defined to work just like `Pod::Simple`'s same-named methods:

```
$parser->parse_file(...)
$parser->parse_string_document(...)
$parser->filter(...)
$parser->parse_from_file(...)
```

For those to work, the Pod-processing subclass of `Pod::Simple::PullParser` has to have defined a `$parser->run` method -- so it is advised that all `Pod::Simple::PullParser` subclasses do so. See the Synopsis above, or the source for `Pod::Simple::RTF`.

Authors of formatter subclasses might find these methods useful to call on a parser object that you haven't started pulling tokens from yet:

```
my $title_string = $parser->get_title
```

This tries to get the title string out of `$parser`, by getting some tokens, and scanning them for the title, and then ungetting them so that you can process the token-stream from the beginning.

For example, suppose you have a document that starts out:

```
=head1 NAME

Hoo::Boy::Wowza -- Stuff B<wow> yeah!
```

`$parser->get_title` on that document will return "Hoo::Boy::Wowza -- Stuff wow yeah!".

In cases where `get_title` can't find the title, it will return empty-string ("").

```
my $title_string = $parser->get_short_title
```

This is just like `get_title`, except that it returns just the modulename, if the title seems to be of the form "SomeModuleName -- description".

For example, suppose you have a document that starts out:

```
=head1 NAME

Hoo::Boy::Wowza -- Stuff B<wow> yeah!
```

then `$parser->get_short_title` on that document will return "Hoo::Boy::Wowza".

But if the document starts out:

```
=head1 NAME

Hooboy, stuff B<wow> yeah!
```

then `$parser->get_short_title` on that document will return "Hooboy, stuff wow yeah!".

If the title can't be found, then `get_short_title` returns empty-string ("").

`$author_name = $parser->get_author`

This works like `get_title` except that it returns the contents of the "=head1 AUTHOR\n\nParagraph...\n" section, assuming that that section isn't terribly long.

(This method tolerates "AUTHORS" instead of "AUTHOR" too.)

`$description_name = $parser->get_description`

This works like `get_title` except that it returns the contents of the "=head1 PARAGRAPH\n\nParagraph...\n" section, assuming that that section isn't terribly long.

`$version_block = $parser->get_version`

This works like `get_title` except that it returns the contents of the "=head1 VERSION\n\n[BIG BLOCK]\n" block. Note that this does NOT return the module's `$VERSION!!`

NOTE

You don't actually *have* to define a `run` method. If you're writing a Pod-formatter class, you should define a `run` just so that users can call `parse_file` etc, but you don't *have* to.

And if you're not writing a formatter class, but are instead just writing a program that does something simple with a `Pod::PullParser` object (and not an object of a subclass), then there's no reason to bother subclassing to add a `run` method.

SEE ALSO

Pod::Simple

Pod::Simple::PullParserToken -- and its subclasses *Pod::Simple::PullParserStartToken*, *Pod::Simple::PullParserTextToken*, and *Pod::Simple::PullParserEndToken*.

HTML::TokeParser, which inspired this.

COPYRIGHT AND DISCLAIMERS

Copyright (c) 2002 Sean M. Burke. All rights reserved.

This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

AUTHOR

Sean M. Burke sburke@cpan.org