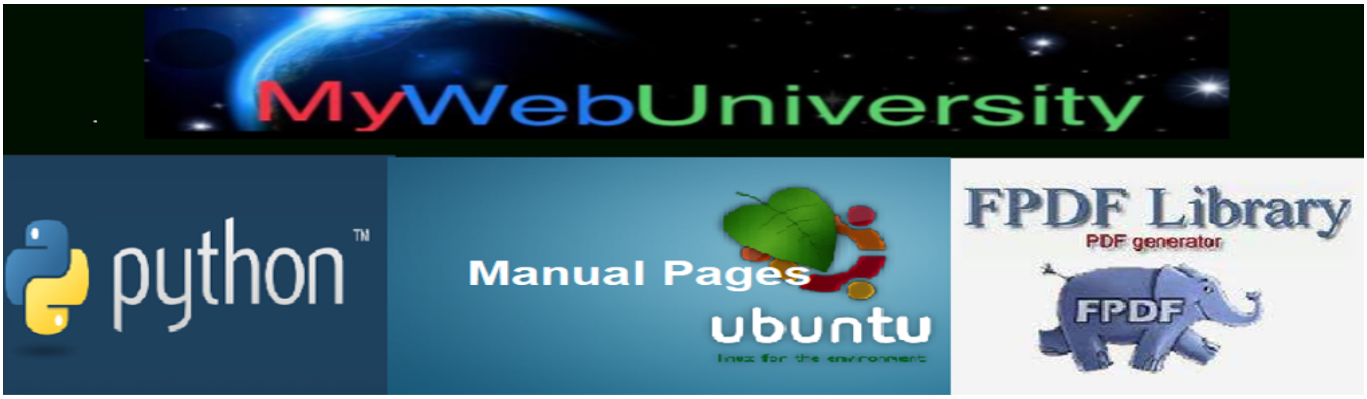


**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

## *Rocky Enterprise Linux 9.2 Manual Pages on command 'HTML::TokeParser.3pm'*

***$ man HTML::TokeParser.3pm***

HTML::TokeParser(3pm) User Contributed Perl Documentation HTML::TokeParser(3pm)

NAME

HTML::TokeParser - Alternative HTML::Parser interface

SYNOPSIS

```
require HTML::TokeParser;
$p = HTML::TokeParser->new("index.html") ||
    die "Can't open: $!";
$p->empty_element_tags(1);  # configure its behaviour
while (my $token = $p->get_token) {
   #...
}
```

DESCRIPTION

The "HTML::TokeParser" is an alternative interface to the "HTML::Parser" class. It is an "HTML::PullParser" subclass with a predeclared set of token types. If you wish the tokens to be reported differently you probably want to use the "HTML::PullParser" directly.

The following methods are available:

```
$p = HTML::TokeParser->new( $filename, %opt );
$p = HTML::TokeParser->new( $filehandle, %opt );
$p = HTML::TokeParser->new( \$document, %opt );
```

The object constructor argument is either a file name, a file handle object, or the complete document to be parsed. Extra options can be provided as key/value pairs and are processed as documented by the base classes.

If the argument is a plain scalar, then it is taken as the name of a file to be opened

and parsed. If the file can't be opened for reading, then the constructor will return "undef" and $! will tell you why it failed.

If the argument is a reference to a plain scalar, then this scalar is taken to be the literal document to parse. The value of this scalar should not be changed before all tokens have been extracted.

Otherwise the argument is taken to be some object that the "HTML::TokeParser" can read() from when it needs more data. Typically it will be a filehandle of some kind. The stream will be read() until EOF, but not closed.

A newly constructed "HTML::TokeParser" differ from its base classes by having the "unbroken_text" attribute enabled by default. See HTML::Parser for a description of this and other attributes that influence how the document is parsed. It is often a good idea to enable "empty_element_tags" behaviour.

Note that the parsing result will likely not be valid if raw undecoded UTF-8 is used as a source. When parsing UTF-8 encoded files turn on UTF-8 decoding:

```
open(my $fh, "<:utf8", "index.html") || die "Can't open 'index.html': $!";
my $p = HTML::TokeParser->new( $fh );
# ...
```

If a $filename is passed to the constructor the file will be opened in raw mode and the parsing result will only be valid if its content is Latin-1 or pure ASCII.

If parsing from an UTF-8 encoded string buffer decode it first:

```
utf8::decode($document);
my $p = HTML::TokeParser->new( \$document );
# ...
```

$p->get_token

This method will return the next token found in the HTML document, or "undef" at the end of the document. The token is returned as an array reference. The first element of the array will be a string denoting the type of this token: "S" for start tag, "E" for end tag, "T" for text, "C" for comment, "D" for declaration, and "PI" for process instructions. The rest of the token array depend on the type like this:

```
["S",  $tag, $attr, $attrseq, $text]
["E",  $tag, $text]
["T",  $text, $is_data]
["C",  $text]
```

["D", $text]

["PI", $token0, $text]

where $attr is a hash reference, $attrseq is an array reference and the rest are plain scalars. The "Argspec" in HTML::Parser explains the details.

$p->unget_token( @tokens )

If you find you have read too many tokens you can push them back, so that they are returned the next time $p->get_token is called.

$p->get_tag

$p->get_tag( @tags )

This method returns the next start or end tag (skipping any other tokens), or "undef" if there are no more tags in the document. If one or more arguments are given, then we skip tokens until one of the specified tag types is found. For example:

$p->get_tag("font", "/font");

will find the next start or end tag for a font-element.

The tag information is returned as an array reference in the same form as for $p->get_token above, but the type code (first element) is missing. A start tag will be returned like this:

[$tag, $attr, $attrseq, $text]

The tagname of end tags are prefixed with "/", i.e. end tag is returned like this:

["/$tag", $text]

$p->get_text

$p->get_text( @endtags )

This method returns all text found at the current position. It will return a zero length string if the next token is not text. Any entities will be converted to their corresponding character.

If one or more arguments are given, then we return all text occurring before the first of the specified tags found. For example:

$p->get_text("p", "br");

will return the text up to either a paragraph of line break element.

The text might span tags that should be textified. This is controlled by the $p->{textify} attribute, which is a hash that defines how certain tags can be treated as text. If the name of a start tag matches a key in this hash then this tag is converted to text. The hash value is used to specify which tag attribute to obtain

the text from. If this tag attribute is missing, then the upper case name of the tag enclosed in brackets is returned, e.g. "[IMG]". The hash value can also be a subroutine reference. In this case the routine is called with the start tag token content as its argument and the return value is treated as the text.

The default $p->{textify} value is:

{img => "alt", applet => "alt"}

This means that <IMG> and <APPLET> tags are treated as text, and that the text to substitute can be found in the ALT attribute.

$p->get_trimmed_text

$p->get_trimmed_text( @endtags )

Same as $p->get_text above, but will collapse any sequences of white space to a single space character. Leading and trailing white space is removed.

$p->get_phrase

This will return all text found at the current position ignoring any phrasal-level tags. Text is extracted until the first non phrasal-level tag. Textification of tags is the same as for get_text(). This method will collapse white space in the same way as get_trimmed_text() does.

The definition of phrasal-level tags is obtained from the HTML::Tagset module.

## EXAMPLES

This example extracts all links from a document. It will print one line for each link, containing the URL and the textual description between the <A>...</A> tags:

```
use HTML::TokeParser;
$p = HTML::TokeParser->new(shift||"index.html");
while (my $token = $p->get_tag("a")) {
    my $url = $token->[1]{href} || "-";
    my $text = $p->get_trimmed_text("/a");
    print "$url\t$text\n";
}
```

This example extract the <TITLE> from the document:

```
use HTML::TokeParser;
$p = HTML::TokeParser->new(shift||"index.html");
if ($p->get_tag("title")) {
    my $title = $p->get_trimmed_text;
```

```
        print "Title: $title\n";
    }
```

SEE ALSO

HTML::PullParser, HTML::Parser

COPYRIGHT

Copyright 1998-2005 Gisle Aas.

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