NAME

MKDoc::XML::Token − XML Token Object

SYNOPSIS

```
my $tokens = MKDoc::XML::Tokenizer->process_data ($some_xml);
foreach my $token (@{$tokens})
{
    print "'" . $token->as_string() . "' is text\n" if (defined $token->text())
    print "'" . $token->as_string() . "' is a self closing tag\n" if (defined $
    print "'" . $token->as_string() . "' is an opening tag\n" if (defined $toke
    print "'" . $token->as_string() . "' is a closing tag\n" if (defined $token-
    print "'" . $token->as_string() . "' is a processing instruction\n" if (def
    print "'" . $token->as_string() . "' is a declaration\n" if (defined $token-
    print "'" . $token->as_string() . "' is a comment\n" if (defined $token->co
    print "'" . $token->as_string() . "' is a tag\n" if (defined $token->tag())
    print "'" . $token->as_string() . "' is a pseudo-tag (NOT text and NOT tag)
    print "'" . $token->as_string() . "' is a leaf token (NOT opening tag)\n" i
}
```

SUMMARY

MKDoc::XML::Token is an object representing an XML token produced by MKDoc::XML::Tokenizer.

It has a set of methods to identify the type of token it is, as well as to help building a parsed tree as in MKDoc::XML::TreeBuilder.

API

my `$token` = **new MKDoc::XML::Token ($string_token);**

Constructs a new MKDoc::XML::Token object.

my `$string_token` = `$token`->*as_string()***;**

Returns the string representation of this token so that:

```
MKDoc::XML::Token->new ($token)->as_string eq $token
```

is a tautology.

my `$node` = `$token`->*leaf()***;**

If this token is not an opening tag, this method will return its corresponding node structure as returned by `$token`->*text()*, `$token`->*tag_self_close()*, etc.

Returns undef otherwise.

my `$node` = `$token`->*pseudotag()***;**

If this token is a comment, declaration or processing instruction, this method will return `$token`->*tag_comment()*, $*token_declaration()* or `$token`->*pi()* resp.

Returns undef otherwise.

my `$node` = `$token`->*tag()***;**

If this token is an opening, closing, or self closing tag, this method will return `$token`->*tag_open()*, `$token`->*tag_close()* or `$token`->*tag_self_close()* resp.

Returns undef otherwise.

my `$node` = `$token`->*comment()***;**

If this token object represents a declaration, the following structure is returned:

```
# this is <!-- I like Pie. Pie is good -->
{
    _tag   => '~comment',
    text   => ' I like Pie. Pie is good ',
}
```

Returns undef otherwise.

my `$node` = `$token`->*declaration()***;**

If this token object represents a declaration, the following structure is returned:

```
# this is <!DOCTYPE foo>
{
    _tag   => '~declaration',
    text   => 'DOCTYPE foo',
}
```

Returns undef otherwise.

my `$node` = `$token->`*pi()*;

If this token object represents a processing instruction, the following structure is returned:

```
# this is <?xml version="1.0" charset="UTF-8"?>
{
    _tag   => '~pi',
    text   => 'xml version="1.0" charset="UTF-8"',
}
```

Returns undef otherwise.

my `$node` = `$token->`*tag_open()*;

If this token object represents an opening tag, the following structure is returned:

```
# this is <aTag foo="bar" baz="buz">
{
    _tag   => 'aTag',
    _open  => 1,
    _close => 0,
    foo    => 'bar',
    baz    => 'buz',
}
```

Returns undef otherwise.

my `$node` = `$token->`*tag_close()*;

If this token object represents a closing tag, the following structure is returned:

```
# this is </aTag>
{
    _tag   => 'aTag',
    _open  => 0,
    _close => 1,
}
```

Returns undef otherwise.

my `$node` = `$token->`*tag_self_close()*;

If this token object represents a self-closing tag, the following structure is returned:

```
# this is <aTag foo="bar" baz="buz" />
{
    _tag   => 'aTag',
    _open  => 1,
    _close => 1,
    foo    => 'bar',
    baz    => 'buz',
}
```

Returns undef otherwise.

my `$node` = `$token->`*text()*;

If this token object represents a piece of text, then this text is returned. Returns undef otherwise. TRAP! `$token->`*text()* returns a false value if this text happens to be '0' or ''. So really you should use:

```
if (defined $token->text()) {
  ... do stuff...
}
```

NOTES

MKDoc::XML::Token works with MKDoc::XML::Tokenizer, which can be used when building a full tree is not necessary. If you need to build a tree, look at MKDoc::XML::TreeBuilder.

AUTHOR

Author: Jean-Michel Hiver

SEE ALSO

MKDoc::XML::Tokenizer MKDoc::XML::TreeBuilder