NAME

XML::XPath::XMLParser − The default XML parsing class that produces a node tree

SYNOPSIS

```
my $parser = XML::XPath::XMLParser->new(
             filename => $self->get_filename,
             xml => $self->get_xml,
             ioref => $self->get_ioref,
             parser => $self->get_parser,
           );
my $root_node = $parser->parse;
```

DESCRIPTION

This module generates a node tree for use as the context node for XPath processing. It aims to be a quick parser, nothing fancy, and yet has to store more information than most parsers. To achieve this I've used array refs everywhere − no hashes. I don't have any performance figures for the speedups achieved, so I make no apologies for anyone not used to using arrays instead of hashes. I think they make good sense here where we know the attributes of each type of node.

Node Structure

All nodes have the same first 2 entries in the array: node_parent and node_pos. The type of the node is determined using the **ref()** function. The node_parent always contains an entry for the parent of the current node − except for the root node which has undef in there. And node_pos is the position of this node in the array that it is in (think: `$node` == `$node`->[node_parent]->[node_children]->[$node->[node_pos]])

Nodes are structured as follows:

Root Node

The root node is just an element node with no parent.

```
[
  undef, # node_parent - check for undef to identify root node
  undef, # node_pos
  undef, # node_prefix
  [ ... ], # node_children (see below)
]
```

Element Node

```
[
  $parent, # node_parent
  <position in current array>, # node_pos
  'xxx', # node_prefix - namespace prefix on this element
  [ ... ], # node_children
  'yyy', # node_name - element tag name
  [ ... ], # node_attribs - attributes on this element
  [ ... ], # node_namespaces - namespaces currently in scope
]
```

Attribute Node

```
[
  $parent, # node_parent - the element node
  <position in current array>, # node_pos
  'xxx', # node_prefix - namespace prefix on this element
  'href', # node_key - attribute name
  'ftp://ftp.com/', # node_value - value in the node
]
```

Namespace Nodes

Each element has an associated set of namespace nodes that are currently in scope. Each namespace node stores a prefix and the expanded name (retrieved from the xmlns:prefix="..." attribute).

```
[
  $parent,
  <pos>,
  'a', # node_prefix - the namespace as it was written as a prefix
  'http://my.namespace.com', # node_expanded - the expanded name.
]
```

Text Nodes

```
[
  $parent,
  <pos>,
  'This is some text' # node_text - the text in the node
]
```

Comment Nodes

```
[
  $parent,
  <pos>,
  'This is a comment' # node_comment
]
```

Processing Instruction Nodes

```
[
  $parent,
  <pos>,
  'target', # node_target
  'data', # node_data
]
```

Usage

If you feel the need to use this module outside of XML::XPath (for example you might use this module directly so that you can cache parsed trees), you can follow the following API:

new

The new method takes either no parameters, or any of the following parameters:

```
filename
xml
parser
ioref
```

This uses the familiar hash syntax, so an example might be:

```
use XML::XPath::XMLParser;

my $parser = XML::XPath::XMLParser->new(filename => 'example.xml');
```

The parameters represent a filename, a string containing XML, an XML::Parser instance and an open filehandle ref respectively. You can also set or get all of these properties using the get_ and set_ functions that have the same name as the property: e.g. get_filename, set_ioref, etc.

parse

The parse method generally takes no parameters, however you are free to pass either an open filehandle reference or an XML string if you so require. The return value is a tree that XML::XPath can use. The parse method will die if there is an error in your XML, so be sure to use perl's exception handling mechanism (eval{};) if you want to avoid this.

parsefile

The parsefile method is identical to**parse()** e xcept it expects a single parameter that is a string naming a file to open and parse. Again it returns a tree and also dies if there are XML errors.

NOTICES