NAME

XML::LibXML::Reader − XML::LibXML::Reader − interface to libxml2 pull parser

SYNOPSIS

```
  use XML::LibXML::Reader;



  my $reader = XML::LibXML::Reader->new(location => "file.xml")
         or die "cannot read file.xml\n";
  while ($reader->read) {
    processNode($reader);
  }



  sub processNode {
      my $reader = shift;
      printf "%d %d %s %d\n", ($reader->depth,
                               $reader->nodeType,
                               $reader->name,
                               $reader->isEmptyElement);
  }
```

or

```
  my $reader = XML::LibXML::Reader->new(location => "file.xml")
         or die "cannot read file.xml\n";
    $reader->preservePattern('//table/tr');
    $reader->finish;
    print $reader->document->toString(1);
```

DESCRIPTION

This is a perl interface to libxml2's pull-parser implementation xmlTextReader *http://xmlsoft.org/html/libxml−xmlreader.html*. This feature requires at least libxml2−2.6.21. Pull-parsers (such as StAX in Java, or XmlReader in C#) use an iterator approach to parse XML documents. They are easier to program than event-based parser (SAX) and much more lightweight than tree-based parser (DOM), which load the complete tree into memory.

The Reader acts as a cursor going forward on the document stream and stopping at each node on the way. At every point, the DOM-like methods of the Reader object allow one to examine the current node (name, namespace, attributes, etc.)

The user's code keeps control of the progress and simply calls the `read()` function repeatedly to progress to the next node in the document order. Other functions provide means for skipping complete sub-trees, or nodes until a specific element, etc.

At every time, only a very limited portion of the document is kept in the memory, which makes the API more memory-efficient than using DOM. However, it is also possible to mix Reader with DOM. At every point the user may copy the current node (optionally expanded into a complete sub-tree) from the processed document to another DOM tree, or to instruct the Reader to collect sub-document in form of a DOM tree consisting of selected nodes.

Reader API also supports namespaces, xml:base, entity handling, and DTD validation. Schema and RelaxNG validation support will probably be added in some later revision of the Perl interface.

The naming of methods compared to libxml2 and C# XmlTextReader has been changed slightly to match the conventions of XML::LibXML. Some functions have been changed or added with respect to the C interface.

CONSTRUCTOR

Depending on the XML source, the Reader object can be created with either of:

```
  my $reader = XML::LibXML::Reader->new( location => "file.xml", ... );
    my $reader = XML::LibXML::Reader->new( string => $xml_string, ... );
    my $reader = XML::LibXML::Reader->new( IO => $file_handle, ... );
    my $reader = XML::LibXML::Reader->new( FD => fileno(STDIN), ... );
    my $reader = XML::LibXML::Reader->new( DOM => $dom, ... );
```

where ... are (optional) reader options described below in "Reader options" or various parser options described in XML::LibXML::Parser. The constructor recognizes the following XML sources:

Source specification

location

Read XML from a local file or URL.

string

Read XML from a string.

IO Read XML a Perl IO filehandle.

FD Read XML from a file descriptor (bypasses Perl I/O layer, only applicable to filehandles for regular files or pipes). Possibly faster than IO.

DOM

Use reader API to walk through a pre-parsed XML::LibXML::Document.

Reader options

encoding => `$encoding`

override document encoding.

RelaxNG => `$rng_schema`

can be used to pass either a XML::LibXML::RelaxNG object or a filename or URL of a RelaxNG schema to the constructor. The schema is then used to validate the document as it is processed.

Schema => `$xsd_schema`

can be used to pass either a XML::LibXML::Schema object or a filename or URL of a W3C XSD schema to the constructor. The schema is then used to validate the document as it is processed.

... the reader further supports various parser options described in XML::LibXML::Parser (specifically those labeled by /reader/).

METHODS CONTROLLING PARSING PROGRESS

read ()

Moves the position to the next node in the stream, exposing its properties.

Returns 1 if the node was read successfully, 0 if there is no more nodes to read, or −1 in case of error

readAttributeValue ()

Parses an attribute value into one or more Text and EntityReference nodes.

Returns 1 in case of success, 0 if the reader was not positioned on an attribute node or all the attribute values have been read, or −1 in case of error.

readState ()

Gets the read state of the reader. Returns the state value, or −1 in case of error. The module exports constants for the Reader states, see STATES below.

depth ()

The depth of the node in the tree, starts at 0 for the root node.

next ()

Skip to the node following the current one in the document order while avoiding the sub-tree if any. Returns 1 if the node was read successfully, 0 if there is no more nodes to read, or −1 in case of error.

nextElement (localname?,nsURI?)

Skip nodes following the current one in the document order until a specific element is reached. The element's name must be equal to a given localname if defined, and its namespace must equal to a given nsURI if defined. Either of the arguments can be undefined (or omitted, in case of the latter or both).

Returns 1 if the element was found, 0 if there is no more nodes to read, or −1 in case of error.

nextPatternMatch (compiled_pattern)

Skip nodes following the current one in the document order until an element matching a given compiled pattern is reached. See XML::LibXML::Pattern for information on compiled patterns. See also the `matchesPattern` method.

Returns 1 if the element was found, 0 if there is no more nodes to read, or −1 in case of error.

skipSiblings ()

Skip all nodes on the same or lower level until the first node on a higher level is reached. In particular, if the current node occurs in an element, the reader stops at the end tag of the parent element, otherwise it stops at a node immediately following the parent node.

Returns 1 if successful, 0 if end of the document is reached, or −1 in case of error.

nextSibling ()

It skips to the node following the current one in the document order while avoiding the sub-tree if any.

Returns 1 if the node was read successfully, 0 if there is no more nodes to read, or −1 in case of error

nextSiblingElement (name?,nsURI?)

Like nextElement but only processes sibling elements of the current node (moving forward using `nextSibling ()` rather than `read ()`, internally).

Returns 1 if the element was found, 0 if there is no more sibling nodes, or −1 in case of error.

finish ()

Skip all remaining nodes in the document, reaching end of the document.

Returns 1 if successful, 0 in case of error.

close ()

This method releases any resources allocated by the current instance and closes any underlying input. It returns 0 on failure and 1 on success. This method is automatically called by the destructor when the reader is forgotten, therefore you do not have to call it directly.

METHODS EXTRACTING INFORMATION

name ()

Returns the qualified name of the current node, equal to (Prefix:)LocalName.

nodeType ()

Returns the type of the current node. See NODE TYPES below.

localName ()

Returns the local name of the node.

prefix ()

Returns the prefix of the namespace associated with the node.

namespaceURI ()

Returns the URI defining the namespace associated with the node.

isEmptyElement ()

Check if the current node is empty, this is a bit bizarre in the sense that <a/> will be considered empty while <a></a> will not.

hasValue ()

Returns true if the node can have a text value.

value ()

Provides the text value of the node if present or undef if not available.

readInnerXml ()

Reads the contents of the current node, including child nodes and markup. Returns a string containing the XML of the node's content, or undef if the current node is neither an element nor attribute, or has no child nodes.

readOuterXml ()

Reads the contents of the current node, including child nodes and markup.

Returns a string containing the XML of the node including its content, or undef if the current node is neither an element nor attribute.

nodePath()

Returns a canonical location path to the current element from the root node to the current node. Namespaced elements are matched by '*', because there is no way to declare prefixes within XPath patterns. Unlike `XML::LibXML::Node::nodePath()`, this function does not provide sibling counts (i.e. instead of e.g. '/a/b[1]' and '/a/b[2]' you get '/a/b' for both matches).

matchesPattern(compiled_pattern)

Returns a true value if the current node matches a compiled pattern. See XML::LibXML::Pattern for information on compiled patterns. See also the `nextPatternMatch` method.

METHODS EXTRACTING DOM NODES

document ()

Provides access to the document tree built by the reader. This function can be used to collect the preserved nodes (see `preserveNode()` and preservePattern).

CAUTION: Never use this function to modify the tree unless reading of the whole document is completed!

copyCurrentNode (deep)

This function is similar a DOM function `copyNode()`. It returns a copy of the currently processed node as a corresponding DOM object. Use deep = 1 to obtain the full sub-tree.

preserveNode ()

This tells the XML Reader to preserve the current node in the document tree. A document tree consisting of the preserved nodes and their content can be obtained using the method `document()` once parsing is finished.

Returns the node or NULL in case of error.

preservePattern (pattern,\%ns_map)

This tells the XML Reader to preserve all nodes matched by the pattern (which is a streaming XPath subset). A document tree consisting of the preserved nodes and their content can be obtained using the method `document()` once parsing is finished.

An optional second argument can be used to provide a HASH reference mapping prefixes used by the XPath to namespace URIs.

The XPath subset available with this function is described at

```
  http://www.w3.org/TR/xmlschema-1/#Selector
```

and matches the production

```
  Path ::= ('.//')? ( Step '/' )* ( Step | '@' NameTest )
```

Returns a positive number in case of success and −1 in case of error

METHODS PROCESSING ATTRIBUTES

attributeCount ()
: Provides the number of attributes of the current node.

hasAttributes ()
: Whether the node has attributes.

getAttribute (name)
: Provides the value of the attribute with the specified qualified name.

  Returns a string containing the value of the specified attribute, or undef in case of error.

getAttributeNs (localName, namespaceURI)
: Provides the value of the specified attribute.

  Returns a string containing the value of the specified attribute, or undef in case of error.

getAttributeNo (no)
: Provides the value of the attribute with the specified index relative to the containing element.

  Returns a string containing the value of the specified attribute, or undef in case of error.

isDefault ()
: Returns true if the current attribute node was generated from the default value defined in the DTD.

moveToAttribute (name)
: Moves the position to the attribute with the specified local name and namespace URI.

  Returns 1 in case of success, −1 in case of error, 0 if not found

moveToAttributeNo (no)
: Moves the position to the attribute with the specified index relative to the containing element.

  Returns 1 in case of success, −1 in case of error, 0 if not found

moveToAttributeNs (localName,namespaceURI)
: Moves the position to the attribute with the specified local name and namespace URI.

  Returns 1 in case of success, −1 in case of error, 0 if not found

moveToFirstAttribute ()
: Moves the position to the first attribute associated with the current node.

  Returns 1 in case of success, −1 in case of error, 0 if not found

moveToNextAttribute ()
: Moves the position to the next attribute associated with the current node.

  Returns 1 in case of success, −1 in case of error, 0 if not found

moveToElement ()
: Moves the position to the node that contains the current attribute node.

  Returns 1 in case of success, −1 in case of error, 0 if not moved

isNamespaceDecl ()
: Determine whether the current node is a namespace declaration rather than a regular attribute.

  Returns 1 if the current node is a namespace declaration, 0 if it is a regular attribute or other type of node, or −1 in case of error.

OTHER METHODS

lookupNamespace (prefix)
: Resolves a namespace prefix in the scope of the current element.

  Returns a string containing the namespace URI to which the prefix maps or undef in case of error.

encoding ()

Returns a string containing the encoding of the document or undef in case of error.

standalone ()

Determine the standalone status of the document being read. Returns 1 if the document was declared to be standalone, 0 if it was declared to be not standalone, or −1 if the document did not specify its standalone status or in case of error.

xmlVersion ()

Determine the XML version of the document being read. Returns a string containing the XML version of the document or undef in case of error.

baseURI ()

Returns the base URI of a given node.

isValid ()

Retrieve the validity status from the parser.

Returns 1 if valid, 0 if no, and −1 in case of error.

xmlLang ()

The xml:lang scope within which the node resides.

lineNumber ()

Provide the line number of the current parsing point.

columnNumber ()

Provide the column number of the current parsing point.

byteConsumed ()

This function provides the current index of the parser relative to the start of the current entity. This function is computed in bytes from the beginning starting at zero and finishing at the size in bytes of the file if parsing a file. The function is of constant cost if the input is UTF−8 but can be costly if run on non−UTF−8 input.

setParserProp (prop => value, ...)

Change the parser processing behaviour by changing some of its internal properties. The following properties are available with this function: ''load_ext_dtd'', ''complete_attributes'', ''validation'', ''expand_entities''.

Since some of the properties can only be changed before any read has been done, it is best to set the parsing properties at the constructor.

Returns 0 if the call was successful, or −1 in case of error

getParserProp (prop)

Get value of an parser internal property. The following property names can be used: ''load_ext_dtd'', ''complete_attributes'', ''validation'', ''expand_entities''.

Returns the value, usually 0 or 1, or −1 in case of error.

DESTRUCTION

XML::LibXML takes care of the reader object destruction when the last reference to the reader object goes out of scope. The document tree is preserved, though, if either of `$reader−>`document or `$reader−>`preserveNode was used and references to the document tree exist.

NODE TYPES

The reader interface provides the following constants for node types (the constant symbols are exported by default or if tag `:types` is used).

```
XML_READER_TYPE_NONE                    => 0
XML_READER_TYPE_ELEMENT                 => 1
XML_READER_TYPE_ATTRIBUTE               => 2
XML_READER_TYPE_TEXT                    => 3
XML_READER_TYPE_CDATA                   => 4
XML_READER_TYPE_ENTITY_REFERENCE        => 5
XML_READER_TYPE_ENTITY                  => 6
XML_READER_TYPE_PROCESSING_INSTRUCTION  => 7
XML_READER_TYPE_COMMENT                 => 8
XML_READER_TYPE_DOCUMENT                => 9
XML_READER_TYPE_DOCUMENT_TYPE           => 10
XML_READER_TYPE_DOCUMENT_FRAGMENT       => 11
XML_READER_TYPE_NOTATION                => 12
XML_READER_TYPE_WHITESPACE              => 13
XML_READER_TYPE_SIGNIFICANT_WHITESPACE  => 14
XML_READER_TYPE_END_ELEMENT             => 15
XML_READER_TYPE_END_ENTITY              => 16
XML_READER_TYPE_XML_DECLARATION         => 17
```

STATES

The following constants represent the values returned by `readState()`. They are exported by default, or if tag `:states` is used:

```
XML_READER_NONE      => -1
XML_READER_START     =>  0
XML_READER_ELEMENT   =>  1
XML_READER_END       =>  2
XML_READER_EMPTY     =>  3
XML_READER_BACKTRACK =>  4
XML_READER_DONE      =>  5
XML_READER_ERROR     =>  6
```

SEE ALSO

XML::LibXML::Pattern for information about compiled patterns.

http://xmlsoft.org/html/libxml−xmlreader.html

http://dotgnu.org/pnetlib−doc/System/Xml/XmlTextReader.html

ORIGINAL IMPLEMENTATION

Heiko Klein, <H.Klein@gmx.net<gt> and Petr Pajas

AUTHORS

Matt Sergeant, Christian Glahn, Petr Pajas

VERSION

2.0134

COPYRIGHT

LICENSE