NAME

XML::LibXML::Node – Abstract Base Class of XML::LibXML Nodes

SYNOPSIS

```
use XML::LibXML;

$name = $node->nodeName;
$node->setNodeName( $newName );
$bool = $node->isSameNode( $other_node );
$bool = $node->isEqual( $other_node );
$num = $node->unique_key;
$content = $node->nodeValue;
$content = $node->textContent;
$type = $node->nodeType;
$node->unbindNode();
$childnode = $node->removeChild( $childnode );
$oldnode = $node->replaceChild( $newNode, $oldNode );
$node->replaceNode($newNode);
$childnode = $node->appendChild( $childnode );
$childnode = $node->addChild( $childnode );
$node = $parent->addNewChild( $nsURI, $name );
$node->addSibling($newNode);
$newnode =$node->cloneNode( $deep );
$parentnode = $node->parentNode;
$nextnode = $node->nextSibling();
$nextnode = $node->nextNonBlankSibling();
$prevnode = $node->previousSibling();
$prevnode = $node->previousNonBlankSibling();
$boolean = $node->hasChildNodes();
$childnode = $node->firstChild;
$childnode = $node->lastChild;
$documentnode = $node->ownerDocument;
$node = $node->getOwner;
$node->setOwnerDocument( $doc );
$node->insertBefore( $newNode, $refNode );
$node->insertAfter( $newNode, $refNode );
@nodes = $node->findnodes( $xpath_expression );
$result = $node->find( $xpath );
print $node->findvalue( $xpath );
$bool = $node->exists( $xpath_expression );
@childnodes = $node->childNodes();
@childnodes = $node->nonBlankChildNodes();
$xmlstring = $node->toString($format,$docencoding);
$c14nstring = $node->toStringC14N();
$c14nstring = $node->toStringC14N($with_comments, $xpath_expression , $xpath_co
$c14nstring = $node->toStringC14N_v1_1();
$c14nstring = $node->toStringC14N_v1_1($with_comments, $xpath_expression , $xpa
$ec14nstring = $node->toStringEC14N();
$ec14nstring = $node->toStringEC14N($with_comments, $xpath_expression, $inclusi
$ec14nstring = $node->toStringEC14N($with_comments, $xpath_expression, $xpath_c
$str = $doc->serialize($format);
$localname = $node->localname;
$nameprefix = $node->prefix;
$uri = $node->namespaceURI();
$boolean = $node->hasAttributes();
```

```
  @attributelist = $node->attributes();
  $URI = $node->lookupNamespaceURI( $prefix );
  $prefix = $node->lookupNamespacePrefix( $URI );
  $node->normalize;
  @nslist = $node->getNamespaces;
  $node->removeChildNodes();
  $strURI = $node->baseURI();
  $node->setBaseURI($strURI);
  $node->nodePath();
  $lineno = $node->line_number();
```

DESCRIPTION

XML::LibXML::Node defines functions that are common to all Node Types. An XML::LibXML::Node should never be created standalone, but as an instance of a high level class such as XML::LibXML::Element or XML::LibXML::Text. The class itself should provide only common functionality. In XML::LibXML each node is part either of a document or a document-fragment. Because of this there is no node without a parent. This may causes confusion with "unbound" nodes.

METHODS

Many functions listed here are extensively documented in the DOM Level 3 specification (<http://www.w3.org/TR/DOM−Level−3−Core/>). Please refer to the specification for extensive documentation.

nodeName

```
  $name = $node->nodeName;
```

Returns the node's name. This function is aware of namespaces and returns the full name of the current node (`prefix:localname`).

Since 1.62 this function also returns the correct DOM names for node types with constant names, namely: #text, #cdata−section, #comment, #document, #document−fragment.

setNodeName

```
  $node->setNodeName( $newName );
```

In very limited situations, it is useful to change a nodes name. In the DOM specification this should throw an error. This Function is aware of namespaces.

isSameNode

```
  $bool = $node->isSameNode( $other_node );
```

returns TRUE (1) if the given nodes refer to the same node structure, otherwise FALSE (0) is returned.

isEqual

```
  $bool = $node->isEqual( $other_node );
```

deprecated version of **isSameNode()**.

NOTE isEqual will change behaviour to follow the DOM specification

unique_key

```
  $num = $node->unique_key;
```

This function is not specified for any DOM level. It returns a key guaranteed to be unique for this node, and to always be the same value for this node. In other words, two node objects return the same key if and only if isSameNode indicates that they are the same node.

The returned key value is useful as a key in hashes.

nodeValue

```
  $content = $node->nodeValue;
```

If the node has any content (such as stored in a `text node`) it can get requested through this function.

NOTE: Element Nodes have no content per definition. To get the text value of an Element use **textContent()** instead!

textContent

```
$content = $node->textContent;
```

this function returns the content of all text nodes in the descendants of the given node as specified in DOM.

nodeType

```
$type = $node->nodeType;
```

Return a numeric value representing the node type of this node. The module XML::LibXML by default exports constants for the node types (see the EXPORT section in the XML::LibXML manual page).

unbindNode

```
$node->unbindNode();
```

Unbinds the Node from its siblings and Parent, but not from the Document it belongs to. If the node is not inserted into the DOM afterwards, it will be lost after the program terminates. From a low level view, the unbound node is stripped from the context it is and inserted into a (hidden) document-fragment.

removeChild

```
$childnode = $node->removeChild( $childnode );
```

This will unbind the Child Node from its parent `$node`. The function returns the unbound node. If `oldNode` is not a child of the given Node the function will fail.

replaceChild

```
$oldnode = $node->replaceChild( $newNode, $oldNode );
```

Replaces the `$oldNode` with the `$newNode`. The `$oldNode` will be unbound from the Node. This function differs from the DOM L2 specification, in the case, if the new node is not part of the document, the node will be imported first.

replaceNode

```
$node->replaceNode($newNode);
```

This function is very similar to **replaceChild()**, but it replaces the node itself rather than a childnode. This is useful if a node found by any XPath function, should be replaced.

appendChild

```
$childnode = $node->appendChild( $childnode );
```

The function will add the `$childnode` to the end of `$node`'s children. The function should fail, if the new childnode is already a child of `$node`. This function differs from the DOM L2 specification, in the case, if the new node is not part of the document, the node will be imported first.

addChild

```
$childnode = $node->addChild( $childnode );
```

As an alternative to **appendChild()** one can use the **addChild()** function. This function is a bit faster, because it avoids all DOM conformity checks. Therefore this function is quite useful if one builds XML documents in memory where the order and ownership (`ownerDocument`) is assured.

addChild() uses libxml2's own **xmlAddChild()** function. Thus it has to be used with extra care: If a text node is added to a node and the node itself or its last childnode is as well a text node, the node to add will be merged with the one already available. The current node will be removed from memory after this action. Because perl is not aware of this action, the perl instance is still available. XML::LibXML will catch the loss of a node and refuse to run any function called on that node.

```
my $t1 = $doc->createTextNode( "foo" );
 my $t2 = $doc->createTextNode( "bar" );
 $t1->addChild( $t2 );       # is OK
 my $val = $t2->nodeValue(); # will fail, script dies
```

Also **addChild()** will not check if the added node belongs to the same document as the node it will be added to. This could lead to inconsistent documents and in more worse cases even to memory violations, if one does not keep track of this issue.

Although this sounds like a lot of trouble, **addChild()** is useful if a document is built from a stream, such as happens sometimes in SAX handlers or filters.

If you are not sure about the source of your nodes, you better stay with **appendChild**(), because this function is more user friendly in the sense of being more error tolerant.

addNewChild

```
$node = $parent->addNewChild( $nsURI, $name );
```

Similar to `addChild()`, this function uses low level libxml2 functionality to provide faster interface for DOM building. ***addNewChild()*** uses `xmlNewChild()` to create a new node on a given parent element.

addNewChild() has two parameters `$nsURI` and `$name`, where `$nsURI` is an (optional) namespace URI. `$name` is the fully qualified element name; **addNewChild()** will determine the correct prefix if necessary.

The function returns the newly created node.

This function is very useful for DOM building, where a created node can be directly associated with its parent. *NOTE* this function is not part of the DOM specification and its use will limit your code to XML::LibXML.

addSibling

```
$node->addSibling($newNode);
```

addSibling() allows adding an additional node to the end of a nodelist, defined by the given node.

cloneNode

```
$newnode =$node->cloneNode( $deep );
```

cloneNode creates a copy of `$node`. When `$deep` is set to 1 (true) the function will copy all child nodes as well. If `$deep` is 0 only the current node will be copied. Note that in case of element, attributes are copied even if `$deep` is 0.

Note that the behavior of this function for `$deep`=0 has changed in 1.62 in order to be consistent with the DOM spec (in older versions attributes and namespace information was not copied for elements).

parentNode

```
$parentnode = $node->parentNode;
```

Returns simply the Parent Node of the current node.

nextSibling

```
$nextnode = $node->nextSibling();
```

Returns the next sibling if any .

nextNonBlankSibling

```
$nextnode = $node->nextNonBlankSibling();
```

Returns the next non-blank sibling if any (a node is blank if it is a Text or CDATA node consisting of whitespace only). This method is not defined by DOM.

previousSibling

```
$prevnode = $node->previousSibling();
```

Analogous to *getNextSibling* the function returns the previous sibling if any.

previousNonBlankSibling

```
$prevnode = $node->previousNonBlankSibling();
```

Returns the previous non-blank sibling if any (a node is blank if it is a Text or CDATA node consisting of whitespace only). This method is not defined by DOM.

hasChildNodes

```
$boolean = $node->hasChildNodes();
```

If the current node has child nodes this function returns TRUE (1), otherwise it returns FALSE (0, not undef).

firstChild

```
$childnode = $node->firstChild;
```

If a node has child nodes this function will return the first node in the child list.

lastChild

```
$childnode = $node->lastChild;
```

If the `$node` has child nodes this function returns the last child node.

ownerDocument

```
$documentnode = $node->ownerDocument;
```

Through this function it is always possible to access the document the current node is bound to.

getOwner

```
$node = $node->getOwner;
```

This function returns the node the current node is associated with. In most cases this will be a document node or a document fragment node.

setOwnerDocument

```
$node->setOwnerDocument( $doc );
```

This function binds a node to another DOM. This method unbinds the node first, if it is already bound to another document.

This function is the opposite calling of XML::LibXML::Document's **adoptNode()** function. Because of this it has the same limitations with Entity References as **adoptNode()**.

insertBefore

```
$node->insertBefore( $newNode, $refNode );
```

The method inserts `$newNode` before `$refNode`. If `$refNode` is undefined, the newNode will be set as the new last child of the parent node. This function differs from the DOM L2 specification, in the case, if the new node is not part of the document, the node will be imported first, automatically.

`$refNode` has to be passed to the function even if it is undefined:

```
$node->insertBefore( $newNode, undef ); # the same as $node->appendChild( $n
 $node->insertBefore( $newNode ); # wrong
```

Note, that the reference node has to be a direct child of the node the function is called on. Also, `$newChild` is not allowed to be an ancestor of the new parent node.

insertAfter

```
$node->insertAfter( $newNode, $refNode );
```

The method inserts `$newNode` after `$refNode`. If `$refNode` is undefined, the newNode will be set as the new last child of the parent node.

Note, that `$refNode` has to be passed explicitly even if it is undef.

findnodes

```
@nodes = $node->findnodes( $xpath_expression );
```

findnodes evaluates the xpath expression (XPath 1.0) on the current node and returns the resulting node set as an array. In scalar context, returns an XML::LibXML::NodeList object.

The xpath expression can be passed either as a string, or as a XML::LibXML::XPathExpression object.

NOTE ON NAMESPACES AND XPATH:

A common mistake about XPath is to assume that node tests consisting of an element name with no prefix match elements in the default namespace. This assumption is wrong – by XPath specification, such node tests can only match elements that are in no (i.e. null) namespace.

So, for example, one cannot match the root element of an XHTML document with `$node->find('/html')` since `'/html'` would only match if the root element `<html>` had no namespace, but all XHTML elements belong to the namespace http://www.w3.org/1999/xhtml. (Note that `xmlns="..."` namespace declarations can also be specified in a DTD, which makes the situation even worse, since the XML document looks as if there was no default namespace).

There are several possible ways to deal with namespaces in XPath:

- The recommended way is to use the XML::LibXML::XPathContext module to define an explicit context for XPath evaluation, in which a document independent prefix-to-namespace mapping can be defined. For example:

  ```
  my $xpc = XML::LibXML::XPathContext->new;
  $xpc->registerNs('x', 'http://www.w3.org/1999/xhtml');
  $xpc->find('/x:html',$node);
  ```

- Another possibility is to use prefixes declared in the queried document (if known). If the document declares a prefix for the namespace in question (and the context node is in the scope of the declaration), `XML::LibXML` allows you to use the prefix in the XPath expression, e.g.:

  ```
  $node->find('/x:html');
  ```

See also XML::LibXML::XPathContext−>findnodes.

find

```
$result = $node->find( $xpath );
```

find evaluates the XPath 1.0 expression using the current node as the context of the expression, and returns the result depending on what type of result the XPath expression had. For example, the XPath "1 * 3 + 52" results in a XML::LibXML::Number object being returned. Other expressions might return an XML::LibXML::Boolean object, or an XML::LibXML::Literal object (a string). Each of those objects uses Perl's overload feature to "do the right thing" in different contexts.

The xpath expression can be passed either as a string, or as a XML::LibXML::XPathExpression object.

See also XML::LibXML::XPathContext−>find.

findvalue

```
print $node->findvalue( $xpath );
```

findvalue is exactly equivalent to:

```
$node->find( $xpath )->to_literal;
```

That is, it returns the literal value of the results. This enables you to ensure that you get a string back from your search, allowing certain shortcuts. This could be used as the equivalent of XSLT's <xsl:value−of select="some_xpath"/>.

See also XML::LibXML::XPathContext−>findvalue.

The xpath expression can be passed either as a string, or as a XML::LibXML::XPathExpression object.

exists
```
$bool = $node->exists( $xpath_expression );
```

This method behaves like *findnodes*, except that it only returns a boolean value (1 if the expression matches a node, 0 otherwise) and may be faster than *findnodes*, because the XPath evaluation may stop early on the first match (this is true for libxml2 >= 2.6.27).

For XPath expressions that do not return node-set, the method returns true if the returned value is a non-zero number or a non-empty string.

childNodes
```
@childnodes = $node->childNodes();
```

childNodes implements a more intuitive interface to the childnodes of the current node. It enables you to pass all children directly to a `map` or `grep`. If this function is called in scalar context, a XML::LibXML::NodeList object will be returned.

nonBlankChildNodes
```
@childnodes = $node->nonBlankChildNodes();
```

This is like *childNodes*, but returns only non-blank nodes (where a node is blank if it is a Text or CDATA node consisting of whitespace only). This method is not defined by DOM.

toString
```
$xmlstring = $node->toString($format,$docencoding);
```

This method is similar to the method `toString` of a XML::LibXML::Document but for a single node. It returns a string consisting of XML serialization of the given node and all its descendants. Unlike `XML::LibXML::Document::toString`, in this case the resulting string is by default a character string (UTF−8 encoded with UTF8 flag on). An optional flag `$format` controls indentation, as in `XML::LibXML::Document::toString`. If the second optional `$docencoding` flag is true, the result will be a byte string in the document encoding (see `XML::LibXML::Document::actualEncoding`).

toStringC14N
```
$c14nstring = $node->toStringC14N();
$c14nstring = $node->toStringC14N($with_comments, $xpath_expression , $xpath
```

The function is similar to **toString()**. Instead of simply serializing the document tree, it transforms it as it is specified in the XML−C14N Specification (see <http://www.w3.org/TR/xml−c14n>). Such transformation is known as canonization.

If `$with_comments` is 0 or not defined, the result-document will not contain any comments that exist in the original document. To include comments into the canonized document, `$with_comments` has to be set to 1.

The parameter `$xpath_expression` defines the nodeset of nodes that should be visible in the resulting document. This can be used to filter out some nodes. One has to note, that only the nodes that are part of the nodeset, will be included into the result-document. Their child-nodes will not exist in the resulting document, unless they are part of the nodeset defined by the xpath expression.

If `$xpath_expression` is omitted or empty, **toStringC14N()** will include all nodes in the given sub-tree, using the following XPath expressions: with comments

```
(. | .//node() | .//@* | .//namespace::*)
```

and without comments

```
(. | .//node() | .//@* | .//namespace::*)[not(self::comment())]
```

An optional parameter `$xpath_context` can be used to pass an XML::LibXML::XPathContext object defining the context for evaluation of `$xpath_expression`. This is useful for mapping namespace prefixes used in the XPath expression to namespace URIs. Note, however, that `$node` will be used as the context node for the evaluation, not the context node of `$xpath_context`!

toStringC14N_v1_1

```
$c14nstring = $node->toStringC14N_v1_1();
$c14nstring = $node->toStringC14N_v1_1($with_comments, $xpath_expression , $
```

This function behaves like **toStringC14N()** except that it uses the "XML_C14N_1_1" constant for canonicalising using the "C14N 1.1 spec".

toStringEC14N

```
$ec14nstring = $node->toStringEC14N();
$ec14nstring = $node->toStringEC14N($with_comments, $xpath_expression, $incl
$ec14nstring = $node->toStringEC14N($with_comments, $xpath_expression, $xpat
```

The function is similar to **toStringC14N()** but follows the XML-EXC-C14N Specification (see <http://www.w3.org/TR/xml-exc-c14n>) for exclusive canonization of XML.

The arguments `$with_comments`, `$xpath_expression`, `$xpath_context` are as in **toStringC14N()**. An ARRAY reference can be passed as the last argument `$inclusive_prefix_list`, listing namespace prefixes that are to be handled in the manner described by the Canonical XML Recommendation (i.e. preserved in the output even if the namespace is not used). C.f. the spec for details.

serialize

```
$str = $doc->serialize($format);
```

An alias for **toString()**. This function was name added to be more consistent with libxml2.

serialize_c14n

An alias for **toStringC14N()**.

serialize_exc_c14n

An alias for **toStringEC14N()**.

localname

```
$localname = $node->localname;
```

Returns the local name of a tag. This is the part behind the colon.

prefix

```
$nameprefix = $node->prefix;
```

Returns the prefix of a tag. This is the part before the colon.

namespaceURI

```
$uri = $node->namespaceURI();
```

returns the URI of the current namespace.

hasAttributes

```
$boolean = $node->hasAttributes();
```

returns 1 (TRUE) if the current node has any attributes set, otherwise 0 (FALSE) is returned.

attributes

```
@attributelist = $node->attributes();
```

This function returns all attributes and namespace declarations assigned to the given node.

Because XML::LibXML does not implement namespace declarations and attributes the same way, it is required to test what kind of node is handled while accessing the functions result.

If this function is called in array context the attribute nodes are returned as an array. In scalar context, the function will return a XML::LibXML::NamedNodeMap object.

lookupNamespaceURI

```
$URI = $node->lookupNamespaceURI( $prefix );
```

Find a namespace URI by its prefix starting at the current node.

lookupNamespacePrefix

```
$prefix = $node->lookupNamespacePrefix( $URI );
```

Find a namespace prefix by its URI starting at the current node.

NOTE Only the namespace URIs are meant to be unique. The prefix is only document related. Also the document might have more than a single prefix defined for a namespace.

normalize

```
$node->normalize;
```

This function normalizes adjacent text nodes. This function is not as strict as libxml2's **xmlTextMerge()** function, since it will not free a node that is still referenced by the perl layer.

getNamespaces

```
@nslist = $node->getNamespaces;
```

If a node has any namespaces defined, this function will return these namespaces. Note, that this will not return all namespaces that are in scope, but only the ones declared explicitly for that node.

Although getNamespaces is available for all nodes, it only makes sense if used with element nodes.

removeChildNodes

```
$node->removeChildNodes();
```

This function is not specified for any DOM level: It removes all childnodes from a node in a single step. Other than the libxml2 function itself (xmlFreeNodeList), this function will not immediately remove the nodes from the memory. This saves one from getting memory violations, if there are nodes still referred to from the Perl level.

baseURI ()

```
$strURI = $node->baseURI();
```

Searches for the base URL of the node. The method should work on both XML and HTML documents even if base mechanisms for these are completely different. It returns the base as defined in RFC 2396 sections "5.1.1. Base URI within Document Content" and "5.1.2. Base URI from the Encapsulating Entity". However it does not return the document base (5.1.3), use method `URI` of `XML::LibXML::Document` for this.

setBaseURI ($strURI)

```
$node->setBaseURI($strURI);
```

This method only does something useful for an element node in an XML document. It sets the xml:base attribute on the node to `$strURI`, which effectively sets the base URI of the node to the same value.

Note: For HTML documents this behaves as if the document was XML which may not be desired, since it does not effectively set the base URI of the node. See RFC 2396 appendix D for an example of how base URI can be specified in HTML.

nodePath

```
$node->nodePath();
```

This function is not specified for any DOM level: It returns a canonical structure based XPath for a given node.

line_number

```
$lineno = $node->line_number();
```

This function returns the line number where the tag was found during parsing. If a node is added to the document the line number is 0. Problems may occur, if a node from one document is passed to another one.

IMPORTANT: Due to limitations in the libxml2 library line numbers greater than 65535 will be returned as 65535. Please see <http://bugzilla.gnome.org/show_bug.cgi?id=325533> for more details.

Note: **line_number()** is special to XML::LibXML and not part of the DOM specification.

If the line_numbers flag of the parser was not activated before parsing, **line_number()** will always return 0.

AUTHORS

Matt Sergeant, Christian Glahn, Petr Pajas

VERSION

2.0134

COPYRIGHT

LICENSE