NAME

XML::LibXML::Element – XML::LibXML Class for Element Nodes

SYNOPSIS

```
use XML::LibXML;
# Only methods specific to Element nodes are listed here,
# see the XML::LibXML::Node manpage for other methods

$node = XML::LibXML::Element->new( $name );
$node->setAttribute( $aname, $avalue );
$node->setAttributeNS( $nsURI, $aname, $avalue );
$avalue = $node->getAttribute( $aname );
$avalue = $node->getAttributeNS( $nsURI, $aname );
$attrnode = $node->getAttributeNode( $aname );
$attrnode = $node->getAttributeNodeNS( $namespaceURI, $aname );
$node->removeAttribute( $aname );
$node->removeAttributeNS( $nsURI, $aname );
$boolean = $node->hasAttribute( $aname );
$boolean = $node->hasAttributeNS( $nsURI, $aname );
@nodes = $node->getChildrenByTagName($tagname);
@nodes = $node->getChildrenByTagNameNS($nsURI,$tagname);
@nodes = $node->getChildrenByLocalName($localname);
@nodes = $node->getElementsByTagName($tagname);
@nodes = $node->getElementsByTagNameNS($nsURI,$localname);
@nodes = $node->getElementsByLocalName($localname);
$node->appendWellBalancedChunk( $chunk );
$node->appendText( $PCDATA );
$node->appendTextNode( $PCDATA );
$node->appendTextChild( $childname , $PCDATA );
$node->setNamespace( $nsURI , $nsPrefix, $activate );
$node->setNamespaceDeclURI( $nsPrefix, $newURI );
$node->setNamespaceDeclPrefix( $oldPrefix, $newPrefix );
```

METHODS

The class inherits from XML::LibXML::Node. The documentation for Inherited methods is not listed here.

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.

new

```
$node = XML::LibXML::Element->new( $name );
```

This function creates a new node unbound to any DOM.

setAttribute

```
$node->setAttribute( $aname, $avalue );
```

This method sets or replaces the node's attribute `$aname` to the value `$avalue`

setAttributeNS

```
$node->setAttributeNS( $nsURI, $aname, $avalue );
```

Namespace-aware version of `setAttribute`, where `$nsURI` is a namespace URI, `$aname` is a qualified name, and `$avalue` is the value. The namespace URI may be null (empty or undefined) in order to create an attribute which has no namespace.

The current implementation differs from DOM in the following aspects

If an attribute with the same local name and namespace URI already exists on the element, but its prefix differs from the prefix of `$aname`, then this function is supposed to change the prefix

(regardless of namespace declarations and possible collisions). However, the current implementation does rather the opposite. If a prefix is declared for the namespace URI in the scope of the attribute, then the already declared prefix is used, disregarding the prefix specified in `$aname`. If no prefix is declared for the namespace, the function tries to declare the prefix specified in `$aname` and dies if the prefix is already taken by some other namespace.

According to DOM Level 2 specification, this method can also be used to create or modify special attributes used for declaring XML namespaces (which belong to the namespace "http://www.w3.org/2000/xmlns/" and have prefix or name "xmlns"). This should work since version 1.61, but again the implementation differs from DOM specification in the following: if a declaration of the same namespace prefix already exists on the element, then changing its value via this method automatically changes the namespace of all elements and attributes in its scope. This is because in libxml2 the namespace URI of an element is not static but is computed from a pointer to a namespace declaration attribute.

getAttribute

```
$avalue = $node->getAttribute( $aname );
```

If `$node` has an attribute with the name `$aname`, the value of this attribute will get returned.

getAttributeNS

```
$avalue = $node->getAttributeNS( $nsURI, $aname );
```

Retrieves an attribute value by local name and namespace URI.

getAttributeNode

```
$attrnode = $node->getAttributeNode( $aname );
```

Retrieve an attribute node by name. If no attribute with a given name exists, `undef` is returned.

getAttributeNodeNS

```
$attrnode = $node->getAttributeNodeNS( $namespaceURI, $aname );
```

Retrieves an attribute node by local name and namespace URI. If no attribute with a given localname and namespace exists, `undef` is returned.

removeAttribute

```
$node->removeAttribute( $aname );
```

The method removes the attribute `$aname` from the node's attribute list, if the attribute can be found.

removeAttributeNS

```
$node->removeAttributeNS( $nsURI, $aname );
```

Namespace version of `removeAttribute`

hasAttribute

```
$boolean = $node->hasAttribute( $aname );
```

This function tests if the named attribute is set for the node. If the attribute is specified, TRUE (1) will be returned, otherwise the return value is FALSE (0).

hasAttributeNS

```
$boolean = $node->hasAttributeNS( $nsURI, $aname );
```

namespace version of `hasAttribute`

getChildrenByTagName

```
@nodes = $node->getChildrenByTagName($tagname);
```

The function gives direct access to all child elements of the current node with a given tagname, where tagname is a qualified name, that is, in case of namespace usage it may consist of a prefix and local name. This function makes things a lot easier if one needs to handle big data sets. A special tagname '*' can be used to match any name.

If this function is called in SCALAR context, it returns the number of elements found.

getChildrenByTagNameNS

```
@nodes = $node->getChildrenByTagNameNS($nsURI,$tagname);
```

Namespace version of `getChildrenByTagName`. A special nsURI '*' matches any namespace URI, in which case the function behaves just like `getChildrenByLocalName`.

If this function is called in SCALAR context, it returns the number of elements found.

getChildrenByLocalName

```
@nodes = $node->getChildrenByLocalName($localname);
```

The function gives direct access to all child elements of the current node with a given local name. It makes things a lot easier if one needs to handle big data sets. A special `localname` '*' can be used to match any local name.

If this function is called in SCALAR context, it returns the number of elements found.

getElementsByTagName

```
@nodes = $node->getElementsByTagName($tagname);
```

This function is part of the spec. It fetches all descendants of a node with a given tagname, where `tagname` is a qualified name, that is, in case of namespace usage it may consist of a prefix and local name. A special `tagname` '*' can be used to match any tag name.

In SCALAR context this function returns an XML::LibXML::NodeList object.

getElementsByTagNameNS

```
@nodes = $node->getElementsByTagNameNS($nsURI,$localname);
```

Namespace version of `getElementsByTagName` as found in the DOM spec. A special `localname` '*' can be used to match any local name and `nsURI` '*' can be used to match any namespace URI.

In SCALAR context this function returns an XML::LibXML::NodeList object.

getElementsByLocalName

```
@nodes = $node->getElementsByLocalName($localname);
```

This function is not found in the DOM specification. It is a mix of getElementsByTagName and getElementsByTagNameNS. It will fetch all tags matching the given local-name. This allows one to select tags with the same local name across namespace borders.

In SCALAR context this function returns an XML::LibXML::NodeList object.

appendWellBalancedChunk

```
$node->appendWellBalancedChunk( $chunk );
```

Sometimes it is necessary to append a string coded XML Tree to a node. *appendWellBalancedChunk* will do the trick for you. But this is only done if the String is `well-balanced`.

*Note that **appendWellBalancedChunk()** is only left for compatibility reasons*. Implicitly it uses

```
my $fragment = $parser->parse_balanced_chunk( $chunk );
 $node->appendChild( $fragment );
```

This form is more explicit and makes it easier to control the flow of a script.

appendText

```
$node->appendText( $PCDATA );
```

alias for **appendTextNode**().

appendTextNode

```
$node->appendTextNode( $PCDATA );
```

This wrapper function lets you add a string directly to an element node.

appendTextChild

```
$node->appendTextChild( $childname , $PCDATA );
```

Somewhat similar with `appendTextNode`: It lets you set an Element, that contains only a `text node` directly by specifying the name and the text content.

setNamespace

```
$node->setNamespace( $nsURI , $nsPrefix, $activate );
```

setNamespace() allows one to apply a namespace to an element. The function takes three parameters: 1. the namespace URI, which is required and the two optional values prefix, which is the namespace prefix, as it should be used in child elements or attributes as well as the additional activate parameter. If prefix is not given, undefined or empty, this function tries to create a declaration of the default namespace.

The activate parameter is most useful: If this parameter is set to FALSE (0), a new namespace declaration is simply added to the element while the element's namespace itself is not altered. Nevertheless, activate is set to TRUE (1) on default. In this case the namespace is used as the node's effective namespace. This means the namespace prefix is added to the node name and if there was a namespace already active for the node, it will be replaced (but its declaration is not removed from the document). A new namespace declaration is only created if necessary (that is, if the element is already in the scope of a namespace declaration associating the prefix with the namespace URI, then this declaration is reused).

The following example may clarify this:

```
my $e1 = $doc->createElement("bar");
 $e1->setNamespace("http://foobar.org", "foo")
```

results

```
<foo:bar xmlns:foo="http://foobar.org"/>
```

while

```
my $e2 = $doc->createElement("bar");
 $e2->setNamespace("http://foobar.org", "foo",0)
```

results only

```
<bar xmlns:foo="http://foobar.org"/>
```

By using `$activate == 0` it is possible to create multiple namespace declarations on a single element.

The function fails if it is required to create a declaration associating the prefix with the namespace URI but the element already carries a declaration with the same prefix but different namespace URI.

setNamespaceDeclURI

```
$node->setNamespaceDeclURI( $nsPrefix, $newURI );
```

EXPERIMENTAL IN 1.61 !

This function manipulates directly with an existing namespace declaration on an element. It takes two parameters: the prefix by which it looks up the namespace declaration and a new namespace URI which replaces its previous value.

It returns 1 if the namespace declaration was found and changed, 0 otherwise.

All elements and attributes (even those previously unbound from the document) for which the namespace declaration determines their namespace belong to the new namespace after the change.

If the new URI is undef or empty, the nodes have no namespace and no prefix after the change.

Namespace declarations once nulled in this way do not further appear in the serialized output (but do remain in the document for internal integrity of libxml2 data structures).

This function is NOT part of any DOM API.

setNamespaceDeclPrefix

```
$node->setNamespaceDeclPrefix( $oldPrefix, $newPrefix );
```

EXPERIMENTAL IN 1.61 !

This function manipulates directly with an existing namespace declaration on an element. It takes two parameters: the old prefix by which it looks up the namespace declaration and a new prefix which is to replace the old one.

The function dies with an error if the element is in the scope of another declaration whose prefix equals to the new prefix, or if the change should result in a declaration with a non-empty prefix but empty namespace URI. Otherwise, it returns 1 if the namespace declaration was found and changed and 0 if not found.

All elements and attributes (even those previously unbound from the document) for which the namespace declaration determines their namespace change their prefix to the new value.

If the new prefix is undef or empty, the namespace declaration becomes a declaration of a default namespace. The corresponding nodes drop their namespace prefix (but remain in the, now default, namespace). In this case the function fails, if the containing element is in the scope of another default namespace declaration.

This function is NOT part of any DOM API.

OVERLOADING

XML::LibXML::Element overloads hash dereferencing to provide access to the element's attributes. For non-namespaced attributes, the attribute name is the hash key, and the attribute value is the hash value. For namespaced attributes, the hash key is qualified with the namespace URI, using Clark notation.

Perl's "tied hash" feature is used, which means that the hash gives you read-write access to the element's attributes. For more information, see XML::LibXML::AttributeHash

AUTHORS

Matt Sergeant, Christian Glahn, Petr Pajas

VERSION

2.0134

COPYRIGHT

LICENSE