Rocky Enterprise Linux 9.2 Manual Pages on command 'XML::LibXML::Document.3pm'

C:\>man XML::LibXML::Document.3pm

XML::LibXML::Document(3pm) User Contributed Perl Documentation XML::LibXML::Document(3pm)

NAME

XML::LibXML::Document - XML::LibXML DOM Document Class

SYNOPSIS

```
use XML::LibXML;
# Only methods specific to Document nodes are listed here,
# see the XML::LibXML::Node manpage for other methods
$dom = XML::LibXML::Document->new( $version, $encoding );
$dom = XML::LibXML::Document->createDocument( $version, $encoding );
$strURI = $doc->URI();
$doc->setURI($strURI);
$strEncoding = $doc->encoding();
$strEncoding = $doc->actualEncoding();
$doc->setEncoding($new_encoding);
$strVersion = $doc->version();
$doc->standalone
$doc->setStandalone($numvalue);
my $compression = $doc->compression;
$doc->setCompression($ziplevel);
$docstring = $dom->toString($format);
$c14nstr = $doc->toStringC14N($comment_flag, $xpath [, $xpath_context ]);
$ec14nstr = $doc->toStringEC14N($comment_flag, $xpath [, $xpath_context ], $inclusive_prefix_list);
```

```
$str = $doc->serialize($format);
$state = $doc->toFile($filename, $format);
$state = $doc->toFH($fh, $format);
$str = $document->toStringHTML();
$str = $document->serialize_html();
$bool = $dom->is_valid();
$dom->validate();
$root = $dom->documentElement();
$dom->setDocumentElement( $root );
$element = $dom->createElement( $nodename );
$element = $dom->createElementNS( $namespaceURI, $nodename );
$text = $dom->createTextNode( $content_text );
$comment = $dom->createComment( $comment_text );
$attrnode = $doc->createAttribute($name [,$value]);
$attrnode = $doc->createAttributeNS( namespaceURI, $name [,$value] );
$fragment = $doc->createDocumentFragment();
$cdata = $dom->createCDATASection( $cdata_content );
my $pi = $doc->createProcessingInstruction( $target, $data );
my $entref = $doc->createEntityReference($refname);
$dtd = $document->createInternalSubset( $rootnode, $public, $system);
$dtd = $document->createExternalSubset( $rootnode_name, $publicId, $systemId);
$document->importNode( $node );
$document->adoptNode( $node );
my $dtd = $doc->externalSubset;
my $dtd = $doc->internalSubset;
$doc->setExternalSubset($dtd);
$doc->setInternalSubset($dtd);
my $dtd = $doc->removeExternalSubset();
my $dtd = $doc->removeInternalSubset();
my @nodelist = $doc->getElementsByTagName($tagname);
my @nodelist = $doc->getElementsByTagNameNS($nsURI,$tagname);
my @nodelist = $doc->getElementsByLocalName($localname);
my $node = $doc->getElementById($id);
```

```
$dom->indexElements();
```

DESCRIPTION

The Document Class is in most cases the result of a parsing process. But sometimes it is necessary to create a Document from scratch. The DOM Document Class provides functions that conform to the DOM Core naming style.

It inherits all functions from XML::LibXML::Node as specified in the DOM specification. This enables access to the nodes besides the root element on document level - a "DTD" for example. The support for these nodes is limited at the moment.

While generally nodes are bound to a document in the DOM concept it is suggested that one should always create a node not bound to any document. There is no need of really including the node to the document, but once the node is bound to a document, it is quite safe that all strings have the correct encoding. If an unbound text node with an ISO encoded string is created (e.g. with $CLASS->new()), the "toString" function may not return the expected result.

To prevent such problems, it is recommended to pass all data to XML::LibXML methods as character strings (i.e. UTF-8 encoded, with the UTF8 flag on).

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.

new

```
$dom = XML::LibXML::Document->new( $version, $encoding );
```

alias for createDocument()

createDocument

```
$dom = XML::LibXML::Document->createDocument( $version, $encoding );
```

The constructor for the document class. As Parameter it takes the version string and (optionally) the encoding string. Simply calling createDocument() will create the document:

```
<?xml version="your version" encoding="your encoding"?>
```

Both parameter are optional. The default value for $version is 1.0, of course.

If the $encoding parameter is not set, the encoding will be left unset, which means UTF-8 is implied.

The call of createDocument() without any parameter will result the following code:

```
<?xml version="1.0"?>
```

Alternatively one can call this constructor directly from the XML::LibXML class level, to avoid some typing. This will not have any effect on the class instance, which is always XML::LibXML::Document.

```
my $document = XML::LibXML->createDocument( "1.0", "UTF-8" );
```

is therefore a shortcut for

```
my $document = XML::LibXML::Document->createDocument( "1.0", "UTF-8" );
```

URI

```
$strURI = $doc->URI();
```

Returns the URI (or filename) of the original document. For documents obtained by parsing a string of a FH without using the URI parsing argument of the corresponding "parse_*" function, the result is a generated string unknown-XYZ where XYZ is some number; for documents created with the constructor "new", the URI is undefined.

The value can be modified by calling "setURI" method on the document node.

setURI

```
$doc->setURI($strURI);
```

Sets the URI of the document reported by the method URI (see also the URI argument to the various "parse_*" functions).

encoding

```
$strEncoding = $doc->encoding();
```

returns the encoding string of the document.

```
my $doc = XML::LibXML->createDocument( "1.0", "ISO-8859-15" );
print $doc->encoding; # prints ISO-8859-15
```

actualEncoding

```
$strEncoding = $doc->actualEncoding();
```

returns the encoding in which the XML will be returned by $doc->toString(). This is usually the original encoding of the document as declared in the XML declaration and returned by $doc->encoding. If the original encoding is not known (e.g. if created in memory or parsed from a XML without a declared encoding), 'UTF-8' is returned.

```
my $doc = XML::LibXML->createDocument( "1.0", "ISO-8859-15" );
print $doc->encoding; # prints ISO-8859-15
```

setEncoding

```
$doc->setEncoding($new_encoding);
```

This method allows one to change the declaration of encoding in the XML declaration of the document. The value also affects the encoding in which the document is serialized to XML by $doc->toString(). Use setEncoding() to remove the encoding declaration.

version

```
$strVersion = $doc->version();
```

returns the version string of the document

getVersion() is an alternative form of this function.

standalone

```
$doc->standalone
```

This function returns the Numerical value of a documents XML declarations standalone attribute. It returns 1 if standalone="yes" was found, 0 if standalone="no" was found and -1 if standalone was not specified (default on creation).

setStandalone

```
$doc->setStandalone($numvalue);
```

Through this method it is possible to alter the value of a documents standalone attribute. Set it to 1 to set standalone="yes", to 0 to set standalone="no" or set it to -1 to remove the standalone attribute from the XML declaration.

compression

```
my $compression = $doc->compression;
```

libxml2 allows reading of documents directly from gzipped files. In this case the compression variable is set to the compression level of that file (0-8). If XML::LibXML parsed a different source or the file wasn't compressed, the returned value will be -1.

setCompression

```
$doc->setCompression($ziplevel);
```

If one intends to write the document directly to a file, it is possible to set the compression level for a given document. This level can be in the range from

0 to 8. If XML::LibXML should not try to compress use -1 (default).

Note that this feature will only work if libxml2 is compiled with zlib support

and toFile() is used for output.

toString

```
$docstring = $dom->toString($format);
```

toString is a DOM serializing function, so the DOM Tree is serialized into an

XML string, ready for output.

IMPORTANT: unlike toString for other nodes, on document nodes this function

returns the XML as a byte string in the original encoding of the document (see

the actualEncoding() method)! This means you can simply do:

```
open my $out_fh, '>', $file;
print {$out_fh} $doc->toString;
```

regardless of the actual encoding of the document. See the section on encodings

in XML::LibXML for more details.

The optional $format parameter sets the indenting of the output. This parameter

is expected to be an "integer" value, that specifies that indentation should be

used. The format parameter can have three different values if it is used:

If $format is 0, than the document is dumped as it was originally parsed

If $format is 1, libxml2 will add ignorable white spaces, so the nodes content

is easier to read. Existing text nodes will not be altered

If $format is 2 (or higher), libxml2 will act as $format == 1 but it add a

leading and a trailing line break to each text node.

libxml2 uses a hard-coded indentation of 2 space characters per indentation

level. This value can not be altered on run-time.

toStringC14N

```
$c14nstr = $doc->toStringC14N($comment_flag, $xpath [, $xpath_context ]);
```

See the documentation in XML::LibXML::Node.

toStringEC14N

```
$ec14nstr = $doc->toStringEC14N($comment_flag, $xpath [, $xpath_context ], $inclusive_prefix_list);
```

See the documentation in XML::LibXML::Node.

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().

toFile

```
$state = $doc->toFile($filename, $format);
```

This function is similar to toString(), but it writes the document directly into a filesystem. This function is very useful, if one needs to store large documents.

The format parameter has the same behaviour as in toString().

toFH

```
$state = $doc->toFH($fh, $format);
```

This function is similar to toString(), but it writes the document directly to a filehandle or a stream. A byte stream in the document encoding is passed to the file handle. Do NOT apply any ":encoding(...)" or ":utf8" PerlIO layer to the filehandle! See the section on encodings in XML::LibXML for more details.

The format parameter has the same behaviour as in toString().

toStringHTML

```
$str = $document->toStringHTML();
```

toStringHTML serialize the tree to a byte string in the document encoding as HTML. With this method indenting is automatic and managed by libxml2 internally.

serialize_html

```
$str = $document->serialize_html();
```

An alias for toStringHTML().

is_valid

```
$bool = $dom->is_valid();
```

Returns either TRUE or FALSE depending on whether the DOM Tree is a valid Document or not.

You may also pass in a XML::LibXML::Dtd object, to validate against an external DTD:

```
if (!$dom->is_valid($dtd)) {
```

```
    warn("document is not valid!");
  }
```

validate

```
  $dom->validate();
```

This is an exception throwing equivalent of is_valid. If the document is not valid it will throw an exception containing the error. This allows you much better error reporting than simply is_valid or not.

Again, you may pass in a DTD object

documentElement

```
  $root = $dom->documentElement();
```

Returns the root element of the Document. A document can have just one root element to contain the documents data.

Optionally one can use getDocumentElement.

setDocumentElement

```
  $dom->setDocumentElement( $root );
```

This function enables you to set the root element for a document. The function supports the import of a node from a different document tree, but does not support a document fragment as $root.

createElement

```
  $element = $dom->createElement( $nodename );
```

This function creates a new Element Node bound to the DOM with the name $nodename.

createElementNS

```
  $element = $dom->createElementNS( $namespaceURI, $nodename );
```

This function creates a new Element Node bound to the DOM with the name $nodename and placed in the given namespace.

createTextNode

```
  $text = $dom->createTextNode( $content_text );
```

As an equivalent of createElement, but it creates a Text Node bound to the DOM.

createComment

```
  $comment = $dom->createComment( $comment_text );
```

As an equivalent of createElement, but it creates a Comment Node bound to the DOM.

createAttribute

$attrnode = $doc->createAttribute($name [,$value]);

Creates a new Attribute node.

createAttributeNS

$attrnode = $doc->createAttributeNS(namespaceURI, $name [,$value]);

Creates an Attribute bound to a namespace.

createDocumentFragment

$fragment = $doc->createDocumentFragment();

This function creates a DocumentFragment.

createCDATASection

$cdata = $dom->createCDATASection($cdata_content);

Similar to createTextNode and createComment, this function creates a

CDataSection bound to the current DOM.

createProcessingInstruction

my $pi = $doc->createProcessingInstruction($target, $data);

create a processing instruction node.

Since this method is quite long one may use its short form createPI().

createEntityReference

my $entref = $doc->createEntityReference($refname);

If a document has a DTD specified, one can create entity references by using

this function. If one wants to add a entity reference to the document, this

reference has to be created by this function.

An entity reference is unique to a document and cannot be passed to other

documents as other nodes can be passed.

NOTE: A text content containing something that looks like an entity reference,

will not be expanded to a real entity reference unless it is a predefined

entity

my $string = "&foo;";

$some_element->appendText($string);

print $some_element->textContent; # prints "&foo;"

createInternalSubset

$dtd = $document->createInternalSubset($rootnode, $public, $system);

This function creates and adds an internal subset to the given document.

Because the function automatically adds the DTD to the document there is no need to add the created node explicitly to the document.

```
my $document = XML::LibXML::Document->new();
 my $dtd      = $document->createInternalSubset( "foo", undef, "foo.dtd" );
```

will result in the following XML document:

```
<?xml version="1.0"?>
 <!DOCTYPE foo SYSTEM "foo.dtd">
```

By setting the public parameter it is possible to set PUBLIC DTDs to a given document. So

```
my $document = XML::LibXML::Document->new();
 my $dtd      = $document->createInternalSubset( "foo", "-//FOO//DTD FOO 0.1//EN", undef );
```

will cause the following declaration to be created on the document:

```
<?xml version="1.0"?>
 <!DOCTYPE foo PUBLIC "-//FOO//DTD FOO 0.1//EN">
```

createExternalSubset

```
$dtd = $document->createExternalSubset( $rootnode_name, $publicId, $systemId);
```

This function is similar to "createInternalSubset()" but this DTD is considered to be external and is therefore not added to the document itself. Nevertheless it can be used for validation purposes.

importNode

```
$document->importNode( $node );
```

If a node is not part of a document, it can be imported to another document. As specified in DOM Level 2 Specification the Node will not be altered or removed from its original document ("$node->cloneNode(1)" will get called implicitly).

NOTE: Don't try to use importNode() to import sub-trees that contain an entity reference - even if the entity reference is the root node of the sub-tree. This will cause serious problems to your program. This is a limitation of libxml2 and not of XML::LibXML itself.

adoptNode

```
$document->adoptNode( $node );
```

If a node is not part of a document, it can be imported to another document. As specified in DOM Level 3 Specification the Node will not be altered but it will removed from its original document.

After a document adopted a node, the node, its attributes and all its descendants belong to the new document. Because the node does not belong to the old document, it will be unlinked from its old location first.

NOTE: Don't try to adoptNode() to import sub-trees that contain entity references - even if the entity reference is the root node of the sub-tree. This will cause serious problems to your program. This is a limitation of libxml2 and not of XML::LibXML itself.

externalSubset

```
my $dtd = $doc->externalSubset;
```

If a document has an external subset defined it will be returned by this function.

NOTE Dtd nodes are no ordinary nodes in libxml2. The support for these nodes in XML::LibXML is still limited. In particular one may not want use common node function on doctype declaration nodes!

internalSubset

```
my $dtd = $doc->internalSubset;
```

If a document has an internal subset defined it will be returned by this function.

NOTE Dtd nodes are no ordinary nodes in libxml2. The support for these nodes in XML::LibXML is still limited. In particular one may not want use common node function on doctype declaration nodes!

setExternalSubset

```
$doc->setExternalSubset($dtd);
```

EXPERIMENTAL!

This method sets a DTD node as an external subset of the given document.

setInternalSubset

```
$doc->setInternalSubset($dtd);
```

EXPERIMENTAL!

This method sets a DTD node as an internal subset of the given document.

removeExternalSubset

```
my $dtd = $doc->removeExternalSubset();
```

EXPERIMENTAL!

If a document has an external subset defined it can be removed from the

document by using this function. The removed dtd node will be returned.

removeInternalSubset

my $dtd = $doc->removeInternalSubset();

EXPERIMENTAL!

If a document has an internal subset defined it can be removed from the document by using this function. The removed dtd node will be returned.

getElementsByTagName

my @nodelist = $doc->getElementsByTagName($tagname);

Implements the DOM Level 2 function

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

getElementsByTagNameNS

my @nodelist = $doc->getElementsByTagNameNS($nsURI,$tagname);

Implements the DOM Level 2 function

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

getElementsByLocalName

my @nodelist = $doc->getElementsByLocalName($localname);

This allows the fetching of all nodes from a given document with the given Localname.

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

getElementById

my $node = $doc->getElementById($id);

Returns the element that has an ID attribute with the given value. If no such element exists, this returns undef.

Note: the ID of an element may change while manipulating the document. For documents with a DTD, the information about ID attributes is only available if DTD loading/validation has been requested. For HTML documents parsed with the HTML parser ID detection is done automatically. In XML documents, all "xml:id" attributes are considered to be of type ID. You can test ID-ness of an attribute node with $attr->isId().

In versions 1.59 and earlier this method was called getElementsById() (plural) by mistake. Starting from 1.60 this name is maintained as an alias only for backward compatibility.

indexElements

```
$dom->indexElements();
```

This function causes libxml2 to stamp all elements in a document with their document position index which considerably speeds up XPath queries for large documents. It should only be used with static documents that won't be further changed by any DOM methods, because once a document is indexed, XPath will always prefer the index to other methods of determining the document order of nodes. XPath could therefore return improperly ordered node-lists when applied on a document that has been changed after being indexed. It is of course possible to use this method to re-index a modified document before using it with XPath again. This function is not a part of the DOM specification.

This function returns number of elements indexed, -1 if error occurred, or -2 if this feature is not available in the running libxml2.

AUTHORS

Matt Sergeant, Christian Glahn, Petr Pajas

VERSION

2.0134

COPYRIGHT

LICENSE