# Document Object Model

# Introduction

The DOM extension allows you to operate on XML documents through the DOM API with PHP 5.

For PHP 4, use DOM XML.

| **Note** |
| --- |
| DOM extension uses UTF-8 encoding. Use utf8_encode() and utf8_decode() to work with texts in ISO-8859-1 encoding or Iconv for other encodings. |

# Installing/Configuring

## Requirements

No external libraries are needed to build this extension.

## Installation

There is no installation needed to use these functions; they are part of the PHP core.

## Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

## Resource Types

This extension has no resource types defined.

# Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

### XML constants

| Constant | Value | Description |
|---|---|---|
| **XML_ELEMENT_NODE** ( integer ) | 1 | Node is a DOMElement |
| **XML_ATTRIBUTE_NODE** ( integer ) | 2 | Node is a DOMAttr |
| **XML_TEXT_NODE** ( integer ) | 3 | Node is a DOMText |
| **XML_CDATA_SECTION_NODE** ( integer ) | 4 | Node is a DOMCharacterData |
| **XML_ENTITY_REF_NODE** ( integer ) | 5 | Node is a DOMEntityReference |
| **XML_ENTITY_NODE** ( integer ) | 6 | Node is a DOMEntity |
| **XML_PI_NODE** ( integer ) | 7 | Node is a DOMProcessingInstruction |
| **XML_COMMENT_NODE** ( integer ) | 8 | Node is a DOMComment |
| **XML_DOCUMENT_NODE** ( integer ) | 9 | Node is a DOMDocument |
| **XML_DOCUMENT_TYPE_NODE** ( integer ) | 10 | Node is a DOMDocumentType |
| **XML_DOCUMENT_FRAG_NODE** ( integer ) | 11 | Node is a DOMDocumentFragment |
| **XML_NOTATION_NODE** ( integer ) | 12 | Node is a DOMNotation |
| **XML_HTML_DOCUMENT_NODE** ( integer ) | 13 | |
| **XML_DTD_NODE** ( integer ) | 14 | |

| | | |
|---|---|---|
| **XML_ELEMENT_DECL_NODE** ( integer ) | 15 | |
| **XML_ATTRIBUTE_DECL_NODE** ( integer ) | 16 | |
| **XML_ENTITY_DECL_NODE** ( integer ) | 17 | |
| **XML_NAMESPACE_DECL_NODE** ( integer ) | 18 | |
| **XML_ATTRIBUTE_CDATA** ( integer ) | 1 | |
| **XML_ATTRIBUTE_ID** ( integer ) | 2 | |
| **XML_ATTRIBUTE_IDREF** ( integer ) | 3 | |
| **XML_ATTRIBUTE_IDREFS** ( integer ) | 4 | |
| **XML_ATTRIBUTE_ENTITY** ( integer ) | 5 | |
| **XML_ATTRIBUTE_NMTOKEN** ( integer ) | 7 | |
| **XML_ATTRIBUTE_NMTOKENS** ( integer ) | 8 | |
| **XML_ATTRIBUTE_ENUMERATION** ( integer ) | 9 | |
| **XML_ATTRIBUTE_NOTATION** ( integer ) | 10 | |

**DOMException constants**

| Constant | Value | Description |
|---|---|---|
| **DOM_INDEX_SIZE_ERR** ( integer ) | 1 | If index or size is negative, or greater than the allowed value. |
| **DOMSTRING_SIZE_ERR** ( integer ) | 2 | If the specified range of text does not fit into a DOMString . |

| | | |
|---|---|---|
| **DOM_HIERARCHY_REQUEST_ERR** ( integer ) | 3 | If any node is inserted somewhere it doesn't belong |
| **DOM_WRONG_DOCUMENT_ERR** ( integer ) | 4 | If a node is used in a different document than the one that created it. |
| **DOM_INVALID_CHARACTER_ERR** ( integer ) | 5 | If an invalid or illegal character is specified, such as in a name. |
| **DOM_NO_DATA_ALLOWED_ERR** ( integer ) | 6 | If data is specified for a node which does not support data. |
| **DOM_NO_MODIFICATION_ALLOWED_ERR** ( integer ) | 7 | If an attempt is made to modify an object where modifications are not allowed. |
| **DOM_NOT_FOUND_ERR** ( integer ) | 8 | If an attempt is made to reference a node in a context where it does not exist. |
| **DOM_NOT_SUPPORTED_ERR** ( integer ) | 9 | If the implementation does not support the requested type of object or operation. |
| **DOM_INUSE_ATTRIBUTE_ERR** ( integer ) | 10 | If an attempt is made to add an attribute that is already in use elsewhere. |
| **DOM_INVALID_STATE_ERR** ( integer ) | 11 | If an attempt is made to use an object that is not, or is no longer, usable. |
| **DOM_SYNTAX_ERR** ( integer ) | 12 | If an invalid or illegal string is specified. |
| **DOM_INVALID_MODIFICATION_ERR** ( integer ) | 13 | If an attempt is made to modify the type of the underlying object. |
| **DOM_NAMESPACE_ERR** ( integer ) | 14 | If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces. |
| **DOM_INVALID_ACCESS_ERR** ( integer ) | 15 | If a parameter or an operation is not supported by the underlying object. |
| **DOM_VALIDATION_ERR** ( | 16 | If a call to a method such as |

| | | |
|---|---|---|
| integer ) | | insertBefore or removeChild would make the Node invalid with respect to "partial validity", this exception would be raised and the operation would not be done. |

# The DOMAttr class

## Introduction

DOMAttr represents an attribute in the DOMElement object.

## Class synopsis

**DOMAttr**

DOMAttr extends DOMNode {

/* Properties */

public readonly string *name*;

public readonly DOMElement *ownerElement*;

public readonly bool *schemaTypeInfo*;

public readonly bool *specified*;

public string *value*;

/* Methods */

**DOMAttr::__construct** ( string `$name` [, string `$value` ] )

bool **DOMAttr::isId** ( void )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*name*
The name of the attribute

*ownerElement*
The element which contains the attribute

*schemaTypeInfo*
Not implemented yet, always is **NULL**

*specified*
Not implemented yet, always is **NULL**

*value*
The value of the attribute

## See Also

- » W3C specification of Attr

# DOMAttr::__construct

DOMAttr::__construct -- Creates a new DOMAttr object

## Description

**DOMAttr::__construct** ( string `$name` [, string `$value` ] )

Creates a new DOMAttr object. This object is read only. It may be appended to a document, but additional nodes may not be appended to this node until the node is associated with a document. To create a writeable node, use DOMDocument::createAttribute.

## Parameters

*`name`*
The tag name of the attribute.

*`value`*
The value of the attribute.

## Examples

**Example #1 - Creating a new DOMAttr**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');
$element = $dom->appendChild(new DOMElement('root'));
$attr = $element->setAttributeNode(new DOMAttr('attr', 'attrvalue'));
echo $dom->saveXML(); /* <?xml version="1.0" encoding="iso-8859-1"?><root
attr="attrvalue" /> */

?>
```

## See Also

- DOMDocument::createAttribute

# DOMAttr::isId

DOMAttr::isId -- Checks if attribute is a defined ID

## Description

bool **DOMAttr::isId** ( void )

This function checks if the attribute is a defined ID.

According to the DOM standard this requires a DTD which defines the attribute ID to be of type ID. You need to validate your document with DOMDocument::validate or *DOMDocument::validateOnParse* before using this function.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Examples

**Example #2 - DOMAttr->isId() Example**

```
<?php

$doc = new DomDocument;

// We need to validate our document before refering to the id
$doc->validateOnParse = true;
$doc->Load('book.xml');

// We retrieve the attribute named id of the chapter element
$attr =
$doc->getElementsByTagName('chapter')->item(0)->getAttributeNode('id');

var_dump($attr->isId()); // bool(true)

?>
```

# The DOMCharacterData class

## Introduction

Represents nodes with character data. No nodes directly correspond to this class, but other nodes do inherit from it.

## Class synopsis

**DOMCharacterData**

DOMCharacterData extends DOMNode {

/* Properties */

public string *data*;

readonly public int *length*;

/* Methods */

void **DOMCharacterData::appendData** ( string `$data` )

void **DOMCharacterData::deleteData** ( int `$offset`, int `$count` )

void **DOMCharacterData::insertData** ( int `$offset`, string `$data` )

void **DOMCharacterData::replaceData** ( int `$offset`, int `$count`, string `$data` )

string **DOMCharacterData::substringData** ( int `$offset`, int `$count` )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*data*
The contents of the node.

*length*
The length of the contents.

## See Also

- » W3C specification of CharacterData

# DOMCharacterData::appendData

DOMCharacterData::appendData -- Append the string to the end of the character data of the node

## Description

void **DOMCharacterData::appendData** ( string `$data` )

Append the string *`data`* to the end of the character data of the node.

## Parameters

*`data`*
: The string to append.

## Return Values

No value is returned.

## See Also

- DOMCharacterData::deleteData
- DOMCharacterData::insertData
- DOMCharacterData::replaceData
- DOMCharacterData::substringData

# DOMCharacterData::deleteData

DOMCharacterData::deleteData -- Remove a range of characters from the node

## Description

void **DOMCharacterData::deleteData** ( int `$offset`, int `$count` )

Deletes *`count`* characters starting from position *`offset`*.

## Parameters

*`offset`*
: The offset from which to start removing.

*`count`*
: The number of characters to delete. If the sum of *`offset`* and *`count`* exceeds the length, then all characters to the end of the data are deleted.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_INDEX_SIZE_ERR**
: Raised if *`offset`* is negative or greater than the number of 16-bit units in data, or if *`count`* is negative.

## See Also

- DOMCharacterData::appendData
- DOMCharacterData::insertData
- DOMCharacterData::replaceData
- DOMCharacterData::substringData

# DOMCharacterData::insertData

DOMCharacterData::insertData -- Insert a string at the specified 16-bit unit offset

## Description

void **DOMCharacterData::insertData** ( int `$offset`, string `$data` )

Inserts string *`data`* at position *`offset`*.

## Parameters

*`offset`*
: The character offset at which to insert.

*`data`*
: The string to insert.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_INDEX_SIZE_ERR**
: Raised if *`offset`* is negative or greater than the number of 16-bit units in data.

## See Also

- DOMCharacterData::appendData
- DOMCharacterData::deleteData
- DOMCharacterData::replaceData
- DOMCharacterData::substringData

# DOMCharacterData::replaceData

DOMCharacterData::replaceData -- Replace a substring within the DOMCharacterData node

## Description

void **DOMCharacterData::replaceData** ( int `$offset`, int `$count`, string `$data` )

Replace *`count`* characters starting from position *`offset`* with *`data`*.

## Parameters

*`offset`*
: The offset from which to start replacing.

*`count`*
: The number of characters to replace. If the sum of *`offset`* and *`count`* exceeds the length, then all characters to the end of the data are replaced.

*`data`*
: The string with which the range must be replaced.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_INDEX_SIZE_ERR**
: Raised if *`offset`* is negative or greater than the number of 16-bit units in data, or if *`count`* is negative.

## See Also

- DOMCharacterData::appendData
- DOMCharacterData::deleteData
- DOMCharacterData::insertData
- DOMCharacterData::substringData

# DOMCharacterData::substringData

DOMCharacterData::substringData -- Extracts a range of data from the node

## Description

string **DOMCharacterData::substringData** ( int `$offset`, int `$count` )

Returns the specified substring.

## Parameters

*`offset`*
: Start offset of substring to extract.

*`count`*
: The number of characters to extract.

## Return Values

The specified substring. If the sum of *`offset`* and *`count`* exceeds the length, then all 16-bit units to the end of the data are returned.

## Errors/Exceptions

**DOM_INDEX_SIZE_ERR**
: Raised if *`offset`* is negative or greater than the number of 16-bit units in data, or if *`count`* is negative.

## See Also

- DOMCharacterData::appendData
- DOMCharacterData::deleteData
- DOMCharacterData::insertData
- DOMCharacterData::replaceData

# The DOMComment class

## Introduction

Represents comment nodes, characters delimited by `<!--` and `-->`.

## Class synopsis

**DOMComment**

DOMComment extends DOMCharacterData {

/* Methods */

__**construct** ( [ string `$value` ] )

/* Inherited methods */

void **DOMCharacterData::appendData** ( string `$data` )

void **DOMCharacterData::deleteData** ( int `$offset`, int `$count` )

void **DOMCharacterData::insertData** ( int `$offset`, string `$data` )

void **DOMCharacterData::replaceData** ( int `$offset`, int `$count`, string `$data` )

string **DOMCharacterData::substringData** ( int `$offset`, int `$count` )
}

## See Also

- » W3C specification of Comment

# DOMComment::__construct

DOMComment::__construct -- Creates a new DOMComment object

## Description

**DOMComment**

__**construct** ( [ string `$value` ] )

Creates a new DOMComment object. This object is read only. It may be appended to a document, but additional nodes may not be appended to this node until the node is associated with a document. To create a writeable node, use DOMDocument::createComment.

## Parameters

*`value`*
The value of the comment.

## Examples

**Example #3 - Creating a new DOMComment**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');
$element = $dom->appendChild(new DOMElement('root'));
$comment = $element->appendChild(new DOMComment('root comment'));
echo $dom->saveXML(); /* <?xml version="1.0"
encoding="iso-8859-1"?><root><!--root comment--></root> */

?>
```

## See Also

- DOMDocument::createComment

# The DOMDocument class

## Introduction

Represents an entire HTML or XML document; serves as the root of the document tree.

## Class synopsis

**DOMDocument**

```
DOMDocument extends DOMNode {

    /*  Properties */

    readonly public string actualEncoding;

    readonly public DOMConfiguration config;

    readonly public DOMDocumentType doctype;

    readonly public DOMElement documentElement;

    public string documentURI;

    public string encoding;

    public bool formatOutput;

    readonly public DOMImplementation implementation;

    public bool preserveWhiteSpace = true;

    public bool recover;

    public bool resolveExternals;

    public bool standalone;

    public bool strictErrorChecking = true;

    public bool substituteEntities;

    public bool validateOnParse = false;
```

public string *version*;

readonly public string *xmlEncoding*;

public bool *xmlStandalone*;

public string *xmlVersion*;

/* Methods */

**DOMDocument::__construct** ( [ string `$version` [, string `$encoding` ] ] )

DOMAttr **DOMDocument::createAttribute** ( string `$name` )

DOMAttr **DOMDocument::createAttributeNS** ( string `$namespaceURI`, string `$qualifiedName` )

DOMCDATASection **DOMDocument::createCDATASection** ( string `$data` )

DOMComment **DOMDocument::createComment** ( string `$data` )

DOMDocumentFragment **DOMDocument::createDocumentFragment** ( void )

DOMElement **DOMDocument::createElement** ( string `$name` [, string `$value` ] )

DOMElement **DOMDocument::createElementNS** ( string `$namespaceURI`, string `$qualifiedName` [, string `$value` ] )

DOMEntityReference **DOMDocument::createEntityReference** ( string `$name` )

DOMProcessingInstruction **DOMDocument::createProcessingInstruction** ( string `$target` [, string `$data` ] )

DOMText **DOMDocument::createTextNode** ( string `$content` )

DOMElement **DOMDocument::getElementById** ( string `$elementId` )

DOMNodeList **DOMDocument::getElementsByTagName** ( string `$name` )

DOMNodeList **DOMDocument::getElementsByTagNameNS** ( string `$namespaceURI`, string `$localName` )

DOMNode **DOMDocument::importNode** ( DOMNode `$importedNode` [, bool `$deep` ] )

mixed **DOMDocument::load** ( string `$filename` [, int `$options` ] )

bool **DOMDocument::loadHTML** ( string `$source` )

bool **DOMDocument::loadHTMLFile** ( string `$filename` )

mixed **DOMDocument::loadXML** ( string `$source` [, int `$options` ] )

void **DOMDocument::normalizeDocument** ( void )

bool **DOMDocument::registerNodeClass** ( string `$baseclass`, string `$extendedclass` )

bool **DOMDocument::relaxNGValidate** ( string `$filename` )

bool **DOMDocument::relaxNGValidateSource** ( string `$source` )

int **DOMDocument::save** ( string `$filename` [, int `$options` ] )

string **DOMDocument::saveHTML** ( void )

int **DOMDocument::saveHTMLFile** ( string `$filename` )

string **DOMDocument::saveXML** ( [ DOMNode `$node` [, int `$options` ] ] )

bool **DOMDocument::schemaValidate** ( string `$filename` )

bool **DOMDocument::schemaValidateSource** ( string `$source` )

bool **DOMDocument::validate** ( void )

int **DOMDocument::xinclude** ( [ int `$options` ] )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )

}

## Properties

*actualEncoding*
: *Deprecated*. Actual encoding of the document, is a readonly equivalent to *encoding*.

*config*
: *Deprecated*. Configuration used when DOMDocument::normalizeDocument() is invoked.

*doctype*
: The Document Type Declaration associated with this document.

*documentElement*
: This is a convenience attribute that allows direct access to the child node that is the document element of the document.

*documentURI*
: The location of the document or **NULL** if undefined.

*encoding*
: Encoding of the document, as specified by the XML declaration. This attribute is not present in the final DOM Level 3 specification, but is the only way of manipulating XML document encoding in this implementation.

*formatOutput*
: Nicely formats output with indentation and extra space.

*implementation*
: The DOMImplementation object that handles this document.

*preserveWhiteSpace*
: Do not remove redundant white space. Default to **TRUE**.

*recover*
: *Proprietary*. Enables recovery mode, i.e. trying to parse non-well formed documents. This attribute is not part of the DOM specification and is specific to libxml.

*resolveExternals*
: Set it to **TRUE** to load external entities from a doctype declaration. This is useful for including character entities in your XML document.

*standalone*
: *Deprecated*. Whether or not the document is standalone, as specified by the XML declaration, corresponds to *xmlStandalone*.

*strictErrorChecking*
: Throws DOMException on errors. Default to **TRUE**.

*substituteEntities*
: *Proprietary*. Whether or not to substitute entities. This attribute is not part of the DOM specification and is specific to libxml.

*validateOnParse*
: Loads and validates against the DTD. Default to **FALSE**.

*version*
: *Deprecated*. Version of XML, corresponds to *xmlVersion*

*xmlEncoding*
: An attribute specifying, as part of the XML declaration, the encoding of this document. This is **NULL** when unspecified or when it is not known, such as when the Document was created in memory.

*xmlStandalone*
: An attribute specifying, as part of the XML declaration, whether this document is standalone. This is **FALSE** when unspecified.

*xmlVersion*
: An attribute specifying, as part of the XML declaration, the version number of this document. If there is no declaration and if this document supports the "XML" feature, the value is "1.0".

## See Also

- » W3C specification for Document

# DOMDocument::__construct

DOMDocument::__construct -- Creates a new DOMDocument object

## Description

**DOMDocument::__construct** ( [ string `$version` [, string `$encoding` ] ] )

Creates a new DOMDocument object.

## Parameters

*`version`*
The version number of the document as part of the XML declaration.

*`encoding`*
The encoding of the document as part of the XML declaration.

## Examples

**Example #4 - Creating a new DOMDocument**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');

echo $dom->saveXML(); /* <?xml version="1.0" encoding="iso-8859-1"?> */

?>
```

## See Also

- DOMImplementation::createDocument

# DOMDocument::createAttribute

DOMDocument::createAttribute -- Create new attribute

## Description

DOMAttr **DOMDocument::createAttribute** ( string `$name` )

This function creates a new instance of class DOMAttr. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*name*
The name of the attribute.

## Return Values

The new DOMAttr or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_INVALID_CHARACTER_ERR**
Raised if *name* contains an invalid character.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createAttributeNS

DOMDocument::createAttributeNS -- Create new attribute node with an associated namespace

## Description

DOMAttr **DOMDocument::createAttributeNS** ( string `$namespaceURI`, string `$qualifiedName` )

This function creates a new instance of class DOMAttr. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`namespaceURI`*
: The URI of the namespace.

*`qualifiedName`*
: The tag name and prefix of the attribute, as *prefix:tagname*.

## Return Values

The new DOMAttr or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_INVALID_CHARACTER_ERR**
: Raised if *`qualifiedName`* contains an invalid character.

**DOM_NAMESPACE_ERR**
: Raised if *`qualifiedName`* is a malformed qualified name, or if *`qualifiedName`* has a prefix and *`namespaceURI`* is **NULL**.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement

- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createCDATASection

DOMDocument::createCDATASection -- Create new cdata node

## Description

DOMCDATASection **DOMDocument::createCDATASection** ( string `$data` )

This function creates a new instance of class DOMCDATASection. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`data`*
The content of the cdata.

## Return Values

The new DOMCDATASection or **FALSE** if an error occured.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createComment

DOMDocument::createComment -- Create new comment node

## Description

DOMComment **DOMDocument::createComment** ( string `$data` )

This function creates a new instance of class DOMComment. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`data`*
The content of the comment.

## Return Values

The new DOMComment or **FALSE** if an error occured.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createDocumentFragment

DOMDocument::createDocumentFragment -- Create new document fragment

## Description

DOMDocumentFragment **DOMDocument::createDocumentFragment** ( void )

This function creates a new instance of class DOMDocumentFragment. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Return Values

The new DOMDocumentFragment or **FALSE** if an error occured.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createElement

DOMDocument::createElement -- Create new element node

## Description

DOMElement **DOMDocument::createElement** ( string `$name` [, string `$value` ] )

This function creates a new instance of class DOMElement. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`name`*
The tag name of the element.

*`value`*
The value of the element. By default, an empty element will be created. You can also set the value later with *DOMElement->nodeValue*.

## Return Values

Returns a new instance of class DOMElement or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_INVALID_CHARACTER_ERR**
Raised if *`name`* contains an invalid character.

## Examples

**Example #5 - Creating a new element and inserting it as root**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');

$element = $dom->createElement('test', 'This is the root element!');

// We insert the new element as root (child of the document)
$dom->appendChild($element);

echo $dom->saveXML();
?>
```

The above example will output:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<test>This is the root element!</test>
```

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createElementNS

DOMDocument::createElementNS -- Create new element node with an associated namespace

## Description

DOMElement **DOMDocument::createElementNS** ( string `$namespaceURI`, string `$qualifiedName` [, string `$value` ] )

This function creates a new element node with an associated namespace. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`namespaceURI`*
The URI of the namespace.

*`qualifiedName`*
The qualified name of the element, as *prefix:tagname*.

*`value`*
The value of the element. By default, an empty element will be created. You can also set the value later with *DOMElement->nodeValue*.

## Return Values

The new DOMElement or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_INVALID_CHARACTER_ERR**
Raised if *`qualifiedName`* contains an invalid character.

**DOM_NAMESPACE_ERR**
Raised if *`qualifiedName`* is a maformed qualified name.

## Examples

**Example #6 - Creating a new element and inserting it as root**

```
<?php
```

```
$dom = new DOMDocument('1.0', 'iso-8859-1');

$element = $dom->createElementNS('http://www.example.com/XFoo', 'xfoo:test',
'This is the root element!');

// We insert the new element as root (child of the document)
$dom->appendChild($element);

echo $dom->saveXML();
?>
```

The above example will output:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xfoo:test xmlns:xfoo="http://www.example.com/XFoo">This is the root
element!</xfoo:test>
```

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createEntityReference

DOMDocument::createEntityReference -- Create new entity reference node

## Description

DOMEntityReference **DOMDocument::createEntityReference** ( string `$name` )

This function creates a new instance of class DOMEntityReference. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*name*
: The content of the entity reference, e.g. the entity reference minus the leading *&*and the trailing*;* characters.

## Return Values

The new DOMEntityReference or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_INVALID_CHARACTER_ERR**
: Raised if *name* contains an invalid character.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createProcessingInstruction
- DOMDocument::createTextNode

# DOMDocument::createProcessingInstruction

DOMDocument::createProcessingInstruction -- Creates new PI node

## Description

DOMProcessingInstruction **DOMDocument::createProcessingInstruction** ( string `$target` [, string `$data` ] )

This function creates a new instance of class DOMProcessingInstruction. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`target`*
: The target of the processing instruction.

*`data`*
: The content of the processing instruction.

## Return Values

The new DOMProcessingInstruction or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_INVALID_CHARACTER_ERR**
: Raised if *`target`* contains an invalid character.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createTextNode

# DOMDocument::createTextNode

DOMDocument::createTextNode -- Create new text node

## Description

DOMText **DOMDocument::createTextNode** ( string `$content` )

This function creates a new instance of class DOMText. This node will not show up in the document unless it is inserted with (e.g.) DOMNode->appendChild().

## Parameters

*`content`*
The content of the text.

## Return Values

The new DOMText or **FALSE** if an error occured.

## See Also

- DOMNode::appendChild
- DOMDocument::createAttribute
- DOMDocument::createAttributeNS
- DOMDocument::createCDATASection
- DOMDocument::createComment
- DOMDocument::createDocumentFragment
- DOMDocument::createElement
- DOMDocument::createElementNS
- DOMDocument::createEntityReference
- DOMDocument::createProcessingInstruction

# DOMDocument::getElementById

DOMDocument::getElementById -- Searches for an element with a certain id

## Description

DOMElement **DOMDocument::getElementById** ( string `$elementId` )

This function is similar to DOMDocument::getElementsByTagName but searches for an element with a given id.

For this function to work, you will need either to set some ID attributes with DOMElement::setIdAttribute or a DTD which defines an attribute to be of type ID. In the later case, you will need to validate your document with DOMDocument::validate or *DOMDocument->validateOnParse* before using this function.

## Parameters

*`elementId`*
The unique id value for an element.

## Return Values

Returns the DOMElement or **NULL** if the element is not found.

## Examples

**Example #7 - DOMDocument->getElementById() Example**

```
<?php

$doc = new DomDocument;

// We need to validate our document before refering to the id
$doc->validateOnParse = true;
$doc->Load('book.xml');

echo "The element whose id is books is: " .
$doc->getElementById('books')->tagName . "\n";

?>
```

The above example will output:

```
The element whose id is books is: chapter
```

## See Also

- DOMDocument::getElementsByTagName

# DOMDocument::getElementsByTagName

DOMDocument::getElementsByTagName -- Searches for all elements with given tag name

## Description

DOMNodeList **DOMDocument::getElementsByTagName** ( string `$name` )

This function returns a new instance of class DOMNodeList containing the elements with a given tag name.

## Parameters

*`name`*
: The name of the tag to match on. The special value * matches all tags.

## Return Values

A new DOMNodeList object containing all the matched elements.

## See Also

- DOMDocument::getElementsByTagNameNS

# DOMDocument::getElementsByTagNameNS

DOMDocument::getElementsByTagNameNS -- Searches for all elements with given tag name in specified namespace

## Description

DOMNodeList **DOMDocument::getElementsByTagNameNS** ( string `$namespaceURI`, string `$localName` )

Returns a DOMNodeList of all elements with a given local name and a namespace URI.

## Parameters

*`namespaceURI`*
: The namespace URI of the elements to match on. The special value * matches all namespaces.

*`localName`*
: The local name of the elements to match on. The special value * matches all local names.

## Return Values

A new DOMNodeList object containing all the matched elements.

## Examples

**Example #8 - Get all the XInclude elements**

```
<?php

$xml = <<<EOD
<?xml version="1.0" ?>
<chapter xmlns:xi="http://www.w3.org/2001/XInclude">
<title>Books of the other guy..</title>
<para>
<xi:include href="book.xml">
 <xi:fallback>
  <error>xinclude: book.xml not found</error>
 </xi:fallback>
</xi:include>
<include>
 This is another namespace
</include>
</para>
</chapter>
```

```
EOD;
$dom = new DOMDocument;

// load the XML string defined above
$dom->loadXML($xml);

foreach ($dom->getElementsByTagNameNS('http://www.w3.org/2001/XInclude',
'*') as $element) {
   echo 'local name: ', $element->localName, ', prefix: ', $element->prefix,
"\n";
}
?>
```

The above example will output:

```
local name: include, prefix: xi
local name: fallback, prefix: xi
```

## See Also

- DOMDocument::getElementsByTagName

# DOMDocument::importNode

DOMDocument::importNode -- Import node into current document

## Description

DOMNode **DOMDocument::importNode** ( DOMNode `$importedNode` [, bool `$deep` ] )

This function returns a copy of the node to import and associates it with the current document.

## Parameters

*`importedNode`*
: The node to import.

*`deep`*
: If set to **TRUE**, this method will recursively import the subtree under the *`importedNode`*.

## Return Values

The copied node or **FALSE**, if it cannot be copied.

## Errors/Exceptions

DOMException is thrown if node cannot be imported.

# DOMDocument::load

DOMDocument::load -- Load XML from a file

## Description

mixed **DOMDocument::load** ( string `$filename` [, int `$options` ] )

Loads an XML document from a file.

| **Warning** |
| --- |
| Unix style paths with forward slashes can cause significant performance degradation on Windows systems; be sure to call realpath() in such a case. |

## Parameters

*`filename`*
The path to the XML document.

*`options`*
Bitwise *OR* of the libxml option constants.

## Return Values

Returns **TRUE** on success or **FALSE** on failure. If called statically, returns a DOMDocument but also causes an E_STRICT warning.

## Errors/Exceptions

If an empty string is passed as the *`filename`* or an empty file is named, a warning will be generated. This warning is not generated by libxml and cannot be handled using libxml's error handling functions.

## Examples

**Example #9 - Creating a Document**

```
<?php
$doc = new DOMDocument();
$doc->load('book.xml');
echo $doc->saveXML();
?>
```

## See Also

- DOMDocument::loadXML
- DOMDocument::save
- DOMDocument::saveXML

# DOMDocument::loadHTML

DOMDocument::loadHTML -- Load HTML from a string

## Description

bool **DOMDocument::loadHTML** ( string `$source` )

The function parses the HTML contained in the string *`source`*. Unlike loading XML, HTML does not have to be well-formed to load. This function may also be called statically to load and create a DOMDocument object. The static invocation may be used when no DOMDocument properties need to be set prior to loading.

## Parameters

*`source`*
The HTML string.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Errors/Exceptions

If an empty string is passed as the *`source`*, a warning will be generated. This warning is not generated by libxml and cannot be handled using libxml's error handling functions.

## Examples

**Example #10 - Creating a Document**

```
<?php
$doc = new DOMDocument();
$doc->loadHTML("<html><body>Test<br></body></html>");
echo $doc->saveHTML();
?>
```

## See Also

- DOMDocument::loadHTMLFile
- DOMDocument::saveHTML

- DOMDocument::saveHTMLFile

# DOMDocument::loadHTMLFile

DOMDocument::loadHTMLFile -- Load HTML from a file

## Description

bool **DOMDocument::loadHTMLFile** ( string `$filename` )

The function parses the HTML document in the file named *`filename`*. Unlike loading XML, HTML does not have to be well-formed to load.

This function may also be called statically to load and create a DOMDocument object. The static invocation may be used when no DOMDocument properties need to be set prior to loading.

## Parameters

*`filename`*
The path to the HTML file.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Errors/Exceptions

If an empty string is passed as the *`filename`* or an empty file is named, a warning will be generated. This warning is not generated by libxml and cannot be handled using libxml's error handling functions.

## Examples

**Example #11 - Creating a Document**

```
<?php
$doc = new DOMDocument();
$doc->loadHTMLFile("filename.html");
echo $doc->saveHTML();
?>
```

## See Also

- DOMDocument::loadHTML
- DOMDocument::saveHTML
- DOMDocument::saveHTMLFile

# DOMDocument::loadXML

DOMDocument::loadXML -- Load XML from a string

## Description

mixed **DOMDocument::loadXML** ( string `$source` [, int `$options` ] )

Loads an XML document from a string.

This method may also be called statically to load and create a DOMDocument object. The static invocation may be used when no DOMDocument properties need to be set prior to loading.

## Parameters

*`source`*
The string containing the XML.

*`options`*
Bitwise *OR* of the libxml option constants.

## Return Values

Returns **TRUE** on success or **FALSE** on failure. If called statically, returns a DOMDocument.

## Errors/Exceptions

If an empty string is passed as the *`source`*, a warning will be generated. This warning is not generated by libxml and cannot be handled using libxml's error handling functions.

## Examples

**Example #12 - Creating a Document**

```
<?php
$doc = new DOMDocument();
$doc->loadXML('<root><node/></root>');
echo $doc->saveXML();
?>
```

**Example #13 - Static invocation of** *loadXML*

```
<?php
$doc = DOMDocument::loadXML('<root><node/></root>');
echo $doc->saveXML();
?>
```

## See Also

- DOMDocument::load
- DOMDocument::save
- DOMDocument::saveXML

# DOMDocument::normalizeDocument

DOMDocument::normalizeDocument -- Normalizes the document

## Description

void **DOMDocument::normalizeDocument** ( void )

This method acts as if you saved and then loaded the document, putting the document in a "normal" form.

## Return Values

No value is returned.

## See Also

- » The DOM Specification
- DOMNode::normalize

# DOMDocument::registerNodeClass

DOMDocument::registerNodeClass -- Register extended class used to create base node type

## Description

bool **DOMDocument::registerNodeClass** ( string `$baseclass`, string `$extendedclass` )

This method allows you to register your own extended DOM class to be used afterward by the PHP DOM extension.

This method is not part of the DOM standard.

## Parameters

*`baseclass`*
: The DOM class that you want to extend. You can find a list of these classes in the chapter introduction. Of course, you won't be able to register a class extending DOMDocument but you can always start your document by instanciating your own extending class.

*`extendedclass`*
: Your extended class name. If **NULL** is provided, any previously registered class extending *`baseclass`* will be removed.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## ChangeLog

| Version | Description |
| --- | --- |
| PHP 5.2.2 | Prior to 5.2.2, a previously registered *`extendedclass`* had to be unregistered before being able to register a new class extending the same *`baseclass`*. |

## Examples

**Example #14 - Adding a new method to DOMElement to ease our code**

```
<?php

class myElement extends DOMElement {
  function appendElement($name) {
     return $this->appendChild(new myElement($name));
  }
}

class myDocument extends DOMDocument {
  function setRoot($name) {
     return $this->appendChild(new myElement($name));
  }
}

$doc = new myDocument();
$doc->registerNodeClass('DOMElement', 'myElement');

// From now on, adding an element to another costs only one method call !
$root = $doc->setRoot('root');
$child = $root->appendElement('child');
$child->setAttribute('foo', 'bar');

echo $doc->saveXML();

?>
```

The above example will output:

```
<?xml version="1.0"?>
<root><child foo="bar"/></root>
```

# DOMDocument::relaxNGValidate

DOMDocument::relaxNGValidate -- Performs relaxNG validation on the document

## Description

bool **DOMDocument::relaxNGValidate** ( string `$filename` )

Performs » relaxNG validation on the document based on the given RNG schema.

## Parameters

*`filename`*
The RNG file.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMDocument::relaxNGValidateSource
- DOMDocument::schemaValidate
- DOMDocument::schemaValidateSource
- DOMDocument::validate

# DOMDocument::relaxNGValidateSource

DOMDocument::relaxNGValidateSource -- Performs relaxNG validation on the document

## Description

bool **DOMDocument::relaxNGValidateSource** ( string `$source` )

Performs » relaxNG validation on the document based on the given RNG source.

## Parameters

*`source`*
A string containing the RNG schema.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMDocument::relaxNGValidate
- DOMDocument::schemaValidate
- DOMDocument::schemaValidateSource
- DOMDocument::validate

# DOMDocument::save

DOMDocument::save -- Dumps the internal XML tree back into a file

## Description

int **DOMDocument::save** ( string `$filename` [, int `$options` ] )

Creates an XML document from the DOM representation. This function is usually called after building a new dom document from scratch as in the example below.

## Parameters

*`filename`*
: The path to the saved XML document.

*`options`*
: Additional Options. Currently only LIBXML_NOEMPTYTAG is supported.

## Return Values

Returns the number of bytes written or **FALSE** if an error occurred.

## ChangeLog

| Version | Description |
|---|---|
| 5.1.0 | Added the *`options`* parameter |

## Examples

**Example #15 - Saving a DOM tree into a file**

```
<?php

$doc = new DOMDocument('1.0');
// we want a nice output
$doc->formatOutput = true;

$root = $doc->createElement('book');
$root = $doc->appendChild($root);
```

```
$title = $doc->createElement('title');
$title = $root->appendChild($title);

$text = $doc->createTextNode('This is the title');
$text = $title->appendChild($text);

echo 'Wrote: ' . $doc->save("/tmp/test.xml") . ' bytes'; // Wrote: 72 bytes

?>
```

## See Also

- DOMDocument::saveXML
- DOMDocument::load
- DOMDocument::loadXML

# DOMDocument::saveHTML

DOMDocument::saveHTML -- Dumps the internal document into a string using HTML formatting

## Description

string **DOMDocument::saveHTML** ( void )

Creates an HTML document from the DOM representation. This function is usually called after building a new dom document from scratch as in the example below.

## Return Values

Returns the HTML, or **FALSE** if an error occurred.

## Examples

**Example #16 - Saving a HTML tree into a string**

```
<?php

$doc = new DOMDocument('1.0');

$root = $doc->createElement('html');
$root = $doc->appendChild($root);

$head = $doc->createElement('head');
$head = $root->appendChild($head);

$title = $doc->createElement('title');
$title = $head->appendChild($title);

$text = $doc->createTextNode('This is the title');
$text = $title->appendChild($text);

echo $doc->saveHTML();

?>
```

## See Also

- DOMDocument::saveHTMLFile
- DOMDocument::loadHTML
- DOMDocument::loadHTMLFile

# DOMDocument::saveHTMLFile

DOMDocument::saveHTMLFile -- Dumps the internal document into a file using HTML formatting

## Description

int **DOMDocument::saveHTMLFile** ( string `$filename` )

Creates an HTML document from the DOM representation. This function is usually called after building a new dom document from scratch as in the example below.

## Parameters

*`filename`*
The path to the saved HTML document.

## Return Values

Returns the number of bytes written or **FALSE** if an error occurred.

## Examples

**Example #17 - Saving a HTML tree into a file**

```
<?php

$doc = new DOMDocument('1.0');
// we want a nice output
$doc->formatOutput = true;

$root = $doc->createElement('html');
$root = $doc->appendChild($root);

$head = $doc->createElement('head');
$head = $root->appendChild($head);

$title = $doc->createElement('title');
$title = $head->appendChild($title);

$text = $doc->createTextNode('This is the title');
$text = $title->appendChild($text);

echo 'Wrote: ' . $doc->saveHTMLFile("/tmp/test.html") . ' bytes'; // Wrote:
129 bytes

?>
```

## See Also

- DOMDocument::saveHTML
- DOMDocument::loadHTML
- DOMDocument::loadHTMLFile

# DOMDocument::saveXML

DOMDocument::saveXML -- Dumps the internal XML tree back into a string

## Description

string **DOMDocument::saveXML** ( [ DOMNode `$node` [, int `$options` ] ] )

Creates an XML document from the DOM representation. This function is usually called after building a new dom document from scratch as in the example below.

## Parameters

*`node`*
Use this parameter to output only a specific node without XML declaration rather than the entire document.

*`options`*
Additional Options. Currently only LIBXML_NOEMPTYTAG is supported.

## Return Values

Returns the XML, or **FALSE** if an error occurred.

## Errors/Exceptions

**DOM_WRONG_DOCUMENT_ERR**
Raised if *`node`* is from another document.

## ChangeLog

| Version | Description |
| --- | --- |
| 5.1.0 | Added the *`options`* parameter |

## Examples

**Example #18 - Saving a DOM tree into a string**

```
<?php

$doc = new DOMDocument('1.0');
// we want a nice output
$doc->formatOutput = true;

$root = $doc->createElement('book');
$root = $doc->appendChild($root);

$title = $doc->createElement('title');
$title = $root->appendChild($title);

$text = $doc->createTextNode('This is the title');
$text = $title->appendChild($text);

echo "Saving all the document:\n";
echo $doc->saveXML() . "\n";

echo "Saving only the title part:\n";
echo $doc->saveXML($title);

?>
```

The above example will output:

```
Saving all the document:
<?xml version="1.0"?>
<book>
 <title>This is the title</title>
</book>

Saving only the title part:
<title>This is the title</title>
```

## See Also

- DOMDocument::save
- DOMDocument::load
- DOMDocument::loadXML

# DOMDocument::schemaValidate

DOMDocument::schemaValidate -- Validates a document based on a schema

## Description

bool **DOMDocument::schemaValidate** ( string `$filename` )

Validates a document based on the given schema file.

## Parameters

*`filename`*
: The path to the schema.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMDocument::schemaValidateSource
- DOMDocument::relaxNGValidate
- DOMDocument::relaxNGValidateSource
- DOMDocument::validate

# DOMDocument::schemaValidateSource

DOMDocument::schemaValidateSource -- Validates a document based on a schema

## Description

bool **DOMDocument::schemaValidateSource** ( string `$source` )

Validates a document based on a schema defined in the given string.

## Parameters

*`source`*
A string containing the schema.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMDocument::schemaValidate
- DOMDocument::relaxNGValidate
- DOMDocument::relaxNGValidateSource
- DOMDocument::validate

# DOMDocument::validate

DOMDocument::validate -- Validates the document based on its DTD

## Description

bool **DOMDocument::validate** ( void )

Validates the document based on its DTD.

You can also use the *validateOnParse* property of DOMDocument to make a DTD validation.

## Return Values

Returns **TRUE** on success or **FALSE** on failure. If the document have no DTD attached, this method will return **FALSE**.

## Examples

**Example #19 - Example of DTD validation**

```
<?php
$dom = new DOMDocument;
$dom->Load('book.xml');
if ($dom->validate()) {
   echo "This document is valid!\n";
}
?>
```

You can also validate your XML file while loading it:

```
<?php
$dom = new DOMDocument;
$dom->validateOnParse = true;
$dom->Load('book.xml');
?>
```

## See Also

- DOMDocument::schemaValidate
- DOMDocument::schemaValidateSource
- DOMDocument::relaxNGValidate
- DOMDocument::relaxNGValidateSource

# DOMDocument::xinclude

DOMDocument::xinclude -- Substitutes XIncludes in a DOMDocument Object

## Description

int **DOMDocument::xinclude** ( [ int `$options` ] )

This method substitutes » XIncludes in a DOMDocument object.

| **Note** |
| --- |
| Due to libxml2 automatically resolving entities, this method will produce unexpected results if the included XML file have an attached DTD. |

## Parameters

*`options`*
libxml parameters. Available since PHP 5.1.0 and Libxml 2.6.7.

## Return Values

Returns the number of XIncludes in the document.

## Examples

**Example #20 - DOMDocument->xinclude() example**

```
<?php

$xml = <<<EOD
<?xml version="1.0" ?>
<chapter xmlns:xi="http://www.w3.org/2001/XInclude">
<title>Books of the other guy..</title>
<para>
 <xi:include href="book.xml">
  <xi:fallback>
   <error>xinclude: book.xml not found</error>
  </xi:fallback>
 </xi:include>
</para>
</chapter>
EOD;

$dom = new DOMDocument;
```

```
// let's have a nice output
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

// load the XML string defined above
$dom->loadXML($xml);

// substitute xincludes
$dom->xinclude();

echo $dom->saveXML();

?>
```

The above example will output something similar to:

```
<?xml version="1.0"?>
<chapter xmlns:xi="http://www.w3.org/2001/XInclude">
 <title>Books of the other guy..</title>
 <para>
   <row xml:base="/home/didou/book.xml">
      <entry>The Grapes of Wrath</entry>
      <entry>John Steinbeck</entry>
      <entry>en</entry>
      <entry>0140186409</entry>
     </row>
   <row xml:base="/home/didou/book.xml">
      <entry>The Pearl</entry>
      <entry>John Steinbeck</entry>
      <entry>en</entry>
      <entry>014017737X</entry>
     </row>
   <row xml:base="/home/didou/book.xml">
      <entry>Samarcande</entry>
      <entry>Amine Maalouf</entry>
      <entry>fr</entry>
      <entry>2253051209</entry>
     </row>
 </para>
</chapter>
```

# The DOMDocumentFragment class

## Class synopsis

**DOMDocumentFragment**

DOMDocumentFragment extends DOMNode {

/* Properties */

/* Methods */

bool **DOMDocumentFragment::appendXML** ( string `$data` )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

# DOMDocumentFragment::appendXML

DOMDocumentFragment::appendXML -- Append raw XML data

## Description

bool **DOMDocumentFragment::appendXML** ( string `$data` )

Appends raw XML data to a DOMDocumentFragment.

This method is not part of the DOM standard. It was created as a simplier approach for appending an XML DocumentFragment in a DOMDocument.

If you want to stick to the standards, you will have to create a temporary DOMDocument with a dummy root and then loop through the child nodes of the root of your XML data to append them.

## Parameters

*`data`*
XML to append.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Examples

**Example #21 - Appending XML data to your document**

```
<?php
$doc = new DOMDocument();
$doc->loadXML("<root/>");
$f = $doc->createDocumentFragment();
$f->appendXML("<foo>text</foo><bar>text2</bar>");
$doc->documentElement->appendChild($f);
echo $doc->saveXML();
?>
```

The above example will output:

```
<?xml version="1.0"?>
<root><foo>text</foo><bar>text2</bar></root>
```

# The DOMDocumentType class

## Introduction

Each DOMDocument has a *doctype* attribute whose value is either **NULL** or a DOMDocumentType object.

## Class synopsis

**DOMDocumentType**

DOMDocumentType extends DOMNode {

/* Properties */

readonly public string *publicId*;

readonly public string *systemId*;

readonly public string *name*;

readonly public DOMNamedNodeMap *entities*;

readonly public DOMNamedNodeMap *notations*;

readonly public string *internalSubset*;

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*publicId*
: The public identifier of the external subset.

*systemId*
: The system identifier of the external subset. This may be an absolute URI or not.

*name*
: The name of DTD; i.e., the name immediately following the *DOCTYPE* keyword.

*entities*
: A DOMNamedNodeMap containing the general entities, both external and internal, declared in the DTD.

*notations*
: A DOMNamedNodeMap containing the notations declared in the DTD.

*internalSubset*
: The internal subset as a string, or null if there is none. This is does not contain the delimiting square brackets.

# The DOMElement class

## Class synopsis

**DOMElement**

DOMElement extends DOMNode {

/* Properties */

readonly public bool *schemaTypeInfo*;

readonly public string *tagName*;

/* Methods */

__**construct** ( string `$name` [, string `$value` [, string `$namespaceURI` ] ] )

string **DOMElement::getAttribute** ( string `$name` )

DOMAttr **DOMElement::getAttributeNode** ( string `$name` )

DOMAttr **DOMElement::getAttributeNodeNS** ( string `$namespaceURI`, string `$localName` )

string **DOMElement::getAttributeNS** ( string `$namespaceURI`, string `$localName` )

DOMNodeList **DOMElement::getElementsByTagName** ( string `$name` )

DOMNodeList **DOMElement::getElementsByTagNameNS** ( string `$namespaceURI`, string `$localName` )

bool **DOMElement::hasAttribute** ( string `$name` )

bool **DOMElement::hasAttributeNS** ( string `$namespaceURI`, string `$localName` )

bool **DOMElement::removeAttribute** ( string `$name` )

bool **DOMElement::removeAttributeNode** ( DOMAttr `$oldnode` )

bool **DOMElement::removeAttributeNS** ( string `$namespaceURI`, string `$localName` )

DOMAttr **DOMElement::setAttribute** ( string `$name`, string `$value` )

DOMAttr **DOMElement::setAttributeNode** ( DOMAttr `$attr` )

DOMAttr **DOMElement::setAttributeNodeNS** ( DOMAttr `$attr` )

void **DOMElement::setAttributeNS** ( string `$namespaceURI`, string `$qualifiedName`, string `$value` )

void **DOMElement::setIdAttribute** ( string `$name`, bool `$isId` )

void **DOMElement::setIdAttributeNode** ( DOMAttr `$attr`, bool `$isId` )

void **DOMElement::setIdAttributeNS** ( string `$namespaceURI`, string `$localName`, bool `$isId` )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*schemaTypeInfo*
Not implemented yet, always return **NULL**

*tagName*
The element name

# DOMElement::__construct

DOMElement::__construct -- Creates a new DOMElement object

## Description

**DOMElement**

**__construct** ( string `$name` [, string `$value` [, string `$namespaceURI` ] ] )

Creates a new DOMElement object. This object is read only. It may be appended to a document, but additional nodes may not be appended to this node until the node is associated with a document. To create a writeable node, use DOMDocument::createElement or DOMDocument::createElementNS.

## Parameters

*name*
: The tag name of the element. When also passing in namespaceURI, the element name may take a prefix to be associated with the URI.

*value*
: The value of the element.

*namespaceURI*
: A namespace URI to create the element within a specific namespace.

## Examples

**Example #22 - Creating a new DOMElement**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');
$element = $dom->appendChild(new DOMElement('root'));
$element_ns = new DOMElement('pr:node1', 'thisvalue', 'http://xyz');
$element->appendChild($element_ns);
echo $dom->saveXML(); /* <?xml version="1.0" encoding="iso-8859-1"?>
<root><pr:node1 xmlns:pr="http://xyz">thisvalue</pr:node1></root> */

?>
```

## See Also

- DOMDocument::createElement
- DOMDocument::createElementNS

# DOMElement::getAttribute

DOMElement::getAttribute -- Returns value of attribute

## Description

string **DOMElement::getAttribute** ( string `$name` )

Gets the value of the attribute with name *`name`* for the current node.

## Parameters

*`name`*
: The name of the attribute.

## Return Values

The value of the attribute, or an empty string if no attribute with the given *`name`* is found.

## See Also

- DOMElement::hasAttribute
- DOMElement::setAttribute
- DOMElement::removeAttribute

# DOMElement::getAttributeNode

DOMElement::getAttributeNode -- Returns attribute node

## Description

DOMAttr **DOMElement::getAttributeNode** ( string `$name` )

Returns the attribute node with name *`name`* for the current element.

## Parameters

*`name`*
: The name of the attribute.

## Return Values

The attribute node.

## See Also

- DOMElement::hasAttribute
- DOMElement::setAttributeNode
- DOMElement::removeAttributeNode

# DOMElement::getAttributeNodeNS

DOMElement::getAttributeNodeNS -- Returns attribute node

## Description

DOMAttr **DOMElement::getAttributeNodeNS** ( string `$namespaceURI`, string `$localName` )

Returns the attribute node in namespace *`namespaceURI`* with local name *`localName`* for the current node.

## Parameters

*`namespaceURI`*
: The namespace URI.

*`localName`*
: The local name.

## Return Values

The attribute node.

## See Also

- DOMElement::hasAttributeNS
- DOMElement::setAttributeNodeNS
- DOMElement::removeAttributeNode

# DOMElement::getAttributeNS

DOMElement::getAttributeNS -- Returns value of attribute

## Description

string **DOMElement::getAttributeNS** ( string `$namespaceURI`, string `$localName` )

Gets the value of the attribute in namespace *`namespaceURI`* with local name *`localName`* for the current node.

## Parameters

*`namespaceURI`*
: The namespace URI.

*`localName`*
: The local name.

## Return Values

The value of the attribute, or an empty string if no attribute with the given *`localName`* and *`namespaceURI`* is found.

## See Also

- DOMElement::hasAttributeNS
- DOMElement::setAttributeNS
- DOMElement::removeAttributeNS

# DOMElement::getElementsByTagName

DOMElement::getElementsByTagName -- Gets elements by tagname

## Description

DOMNodeList **DOMElement::getElementsByTagName** ( string `$name` )

This function returns a new instance of the class DOMNodeList of all descendant elements with a given tag *`name`*, in the order in which they are encountered in a preorder traversal of this element tree.

## Parameters

*`name`*
: The tag name. Use * to return all elements within the element tree.

## Return Values

This function returns a new instance of the class DOMNodeList of all matched elements.

## See Also

- DOMElement::getElementsByTagNameNS

# DOMElement::getElementsByTagNameNS

DOMElement::getElementsByTagNameNS -- Get elements by namespaceURI and localName

## Description

DOMNodeList **DOMElement::getElementsByTagNameNS** ( string `$namespaceURI`, string `$localName` )

This function fetch all the descendant elements with a given *`localName`* and *`namespaceURI`*.

## Parameters

*`namespaceURI`*
: The namespace URI.

*`localName`*
: The local name. Use * to return all elements within the element tree.

## Return Values

This function returns a new instance of the class DOMNodeList of all matched elements in the order in which they are encountered in a preorder traversal of this element tree.

## See Also

- DOMElement::getElementsByTagName

# DOMElement::hasAttribute

DOMElement::hasAttribute -- Checks to see if attribute exists

## Description

bool **DOMElement::hasAttribute** ( string `$name` )

Indicates whether attribute named *`name`* exists as a member of the element.

## Parameters

*`name`*
: The attribute name.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMElement::hasAttributeNS
- DOMElement::getAttribute
- DOMElement::setAttribute
- DOMElement::removeAttribute

# DOMElement::hasAttributeNS

DOMElement::hasAttributeNS -- Checks to see if attribute exists

## Description

bool **DOMElement::hasAttributeNS** ( string `$namespaceURI`, string `$localName` )

Indicates whether attribute in namespace *`namespaceURI`* named *`localName`* exists as a member of the element.

## Parameters

*`namespaceURI`*
: The namespace URI.

*`localName`*
: The local name.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMElement::hasAttribute
- DOMElement::getAttributeNS
- DOMElement::setAttributeNS
- DOMElement::removeAttributeNS

# DOMElement::removeAttribute

DOMElement::removeAttribute -- Removes attribute

## Description

bool **DOMElement::removeAttribute** ( string `$name` )

Removes attribute named *`name`* from the element.

## Parameters

*`name`*
: The name of the attribute.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

## See Also

- DOMElement::hasAttribute
- DOMElement::getAttribute
- DOMElement::setAttribute

# DOMElement::removeAttributeNode

DOMElement::removeAttributeNode -- Removes attribute

## Description

bool **DOMElement::removeAttributeNode** ( DOMAttr `$oldnode` )

Removes attribute *`oldnode`* from the element.

## Parameters

*`oldnode`*
The attribute node.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
Raised if the node is readonly.

**DOM_NOT_FOUND_ERROR**
Raised if *`oldnode`* is not an attribute of the element.

## See Also

- DOMElement::hasAttribute
- DOMElement::getAttributeNode
- DOMElement::setAttributeNode

# DOMElement::removeAttributeNS

DOMElement::removeAttributeNS -- Removes attribute

## Description

bool **DOMElement::removeAttributeNS** ( string `$namespaceURI`, string `$localName` )

Removes attribute is namespace *`namespaceURI`* named *`localName`* from the element.

## Parameters

*`namespaceURI`*
: The namespace URI.

*`localName`*
: The local name.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

## See Also

- DOMElement::hasAttributeNS
- DOMElement::getAttributeNS
- DOMElement::setAttributeNS

# DOMElement::setAttribute

DOMElement::setAttribute -- Adds new attribute

## Description

DOMAttr **DOMElement::setAttribute** ( string `$name`, string `$value` )

Sets an attribute with name *`name`* to the given value. If the attribute does not exist, it will be created.

## Parameters

*`name`*
The name of the attribute.

*`value`*
The value of the attribute.

## Return Values

The new DOMAttr or **FALSE** if an error occured.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
Raised if the node is readonly.

## Examples

**Example #23 - Setting an attribute**

```
<?php
$doc = new DOMDocument("1.0");
$node = $doc->createElement("para");
$newnode = $doc->appendChild($node);
$newnode->setAttribute("align", "left");
?>
```

## See Also

- DOMElement::hasAttribute
- DOMElement::getAttribute
- DOMElement::removeAttribute

# DOMElement::setAttributeNode

DOMElement::setAttributeNode -- Adds new attribute node to element

## Description

DOMAttr **DOMElement::setAttributeNode** ( DOMAttr `$attr` )

Adds new attribute node *`attr`* to element.

## Parameters

*`attr`*
: The attribute node.

## Return Values

Returns old node if the attribute has been replaced or **NULL**.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

## See Also

- DOMElement::hasAttribute
- DOMElement::getAttributeNode
- DOMElement::removeAttributeNode

# DOMElement::setAttributeNodeNS

DOMElement::setAttributeNodeNS -- Adds new attribute node to element

## Description

DOMAttr **DOMElement::setAttributeNodeNS** ( DOMAttr `$attr` )

Adds new attribute node *`attr`* to element.

## Parameters

*`name`*
: The attribute node.

## Return Values

Returns the old node if the attribute has been replaced.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

## See Also

- DOMElement::hasAttributeNS
- DOMElement::getAttributeNodeNS
- DOMElement::removeAttributeNode

# DOMElement::setAttributeNS

DOMElement::setAttributeNS -- Adds new attribute

## Description

void **DOMElement::setAttributeNS** ( string `$namespaceURI`, string `$qualifiedName`, string `$value` )

Sets an attribute with namespace *`namespaceURI`* and name *`name`* to the given value. If the attribute does not exist, it will be created.

## Parameters

*`namespaceURI`*
: The namespace URI.

*`qualifiedName`*
: The qualified name of the attribute, as *prefix:tagname*.

*`value`*
: The value of the attribute.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

**DOM_NAMESPACE_ERR**
: Raised if *`qualifiedName`* is a malformed qualified name, or if *`qualifiedName`* has a prefix and *`namespaceURI`* is **NULL**.

## See Also

- DOMElement::hasAttributeNS
- DOMElement::getAttributeNS
- DOMElement::removeAttributeNS

# DOMElement::setIdAttribute

DOMElement::setIdAttribute -- Declares the attribute specified by name to be of type ID

## Description

void **DOMElement::setIdAttribute** ( string `$name`, bool `$isId` )

Declares the attribute *`name`* to be of type ID.

## Parameters

*`name`*
: The name of the attribute.

*`isId`*
: Set it to **TRUE** if you want *`name`* to be of type ID, **FALSE** otherwise.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

**DOM_NOT_FOUND**
: Raised if *`name`* is not an attribute of this element.

## See Also

- DOMDocument::getElementById
- DOMElement::setIdAttributeNode
- DOMElement::setIdAttributeNS

# DOMElement::setIdAttributeNode

DOMElement::setIdAttributeNode -- Declares the attribute specified by node to be of type ID

## Description

void **DOMElement::setIdAttributeNode** ( DOMAttr `$attr`, bool `$isId` )

Declares the attribute specified by *`attr`* to be of type ID.

## Parameters

*`attr`*
: The attribute node.

*`isId`*
: Set it to **TRUE** if you want *`name`* to be of type ID, **FALSE** otherwise.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

**DOM_NOT_FOUND**
: Raised if *`name`* is not an attribute of this element.

## See Also

- DOMDocument::getElementById
- DOMElement::setIdAttribute
- DOMElement::setIdAttributeNS

# DOMElement::setIdAttributeNS

DOMElement::setIdAttributeNS -- Declares the attribute specified by local name and namespace URI to be of type ID

## Description

void **DOMElement::setIdAttributeNS** ( string `$namespaceURI`, string `$localName`, bool `$isId` )

Declares the attribute specified by *`localName`* and *`namespaceURI`* to be of type ID.

## Parameters

*`namespaceURI`*
: The namespace URI of the attribute.

*`localName`*
: The local name of the attribute, as *prefix:tagname*.

*`isId`*
: Set it to **TRUE** if you want *`name`* to be of type ID, **FALSE** otherwise.

## Return Values

No value is returned.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
: Raised if the node is readonly.

**DOM_NOT_FOUND**
: Raised if *`name`* is not an attribute of this element.

## See Also

- DOMDocument::getElementById
- DOMElement::setIdAttribute
- DOMElement::setIdAttributeNode

# The DOMEntity class

## Introduction

This interface represents a known entity, either parsed or unparsed, in an XML document.

## Class synopsis

**DOMEntity**

DOMEntity extends DOMNode {

/* Properties */

readonly public string *publicId*;

readonly public string *systemId*;

readonly public string *notationName*;

public string *actualEncoding*;

readonly public string *encoding*;

readonly public string *version*;

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*publicId*
The public identifier associated with the entity if specified, and **NULL** otherwise.

*systemId*
The system identifier associated with the entity if specified, and **NULL** otherwise. This may be an absolute URI or not.

*notationName*
For unparsed entities, the name of the notation for the entity. For parsed entities, this is **NULL**.

*actualEncoding*
An attribute specifying the encoding used for this entity at the time of parsing, when it is an external parsed entity. This is **NULL** if it an entity from the internal subset or if it is not known.

*encoding*
An attribute specifying, as part of the text declaration, the encoding of this entity, when it is an external parsed entity. This is **NULL** otherwise.

*version*
An attribute specifying, as part of the text declaration, the version number of this entity, when it is an external parsed entity. This is **NULL** otherwise.

# The DOMEntityReference class

## Class synopsis

**DOMEntityReference**

DOMEntityReference extends DOMNode {

/* Properties */

/* Methods */

__**construct** ( string `$name` )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

# DOMEntityReference::__construct

DOMEntityReference::__construct -- Creates a new DOMEntityReference object

## Description

**DOMEntityReference**

**__construct** ( string `$name` )

Creates a new DOMEntityReference object.

## Parameters

*name*
The name of the entity reference.

## Examples

**Example #24 - Creating a new DOMEntityReference**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');
$element = $dom->appendChild(new DOMElement('root'));
$entity = $element->appendChild(new DOMEntityReference('nbsp'));
echo $dom->saveXML(); /* <?xml version="1.0"
encoding="iso-8859-1"?><root> </root> */

?>
```

## See Also

- DOMDocument::createEntityReference

# The DOMException class

## Introduction

DOM operations raise exceptions under particular circumstances, i.e., when an operation is impossible to perform for logical reasons.

See also Exceptions.

## Class synopsis

**DOMException**

```
DOMException extends Exception {

    /*  Properties */

    readonly public int code;
}
```

## Properties

*code*
: An integer indicating the type of error generated

# The DOMImplementation class

## Introduction

The DOMImplementation interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

## Class synopsis

**DOMImplementation**

DOMImplementation {

/* Properties */

/* Methods */

__**construct** ( void )

DOMDocument **DOMImplementation::createDocument** ( [ string `$namespaceURI` [, string `$qualifiedName` [, DOMDocumentType `$doctype` ] ] ] )

DOMDocumentType **DOMImplementation::createDocumentType** ( [ string `$qualifiedName` [, string `$publicId` [, string `$systemId` ] ] ] )

bool **DOMImplementation::hasFeature** ( string `$feature`, string `$version` )
}

# DOMImplementation::__construct

DOMImplementation::__construct -- Creates a new DOMImplementation object

## Description

| **DOMImplementation** |
| --- |

__**construct** ( void )

Creates a new DOMImplementation object.

# DOMImplementation::createDocument

DOMImplementation::createDocument -- Creates a DOMDocument object of the specified type with its document element

## Description

DOMDocument **DOMImplementation::createDocument** ( [ string `$namespaceURI` [, string `$qualifiedName` [, DOMDocumentType `$doctype` ] ] ] )

Creates a DOMDocument object of the specified type with its document element.

## Parameters

*`namespaceURI`*
: The namespace URI of the document element to create.

*`qualifiedName`*
: The qualified name of the document element to create.

*`doctype`*
: The type of document to create or **NULL**.

## Return Values

A new DOMDocument object. If *`namespaceURI`*, *`qualifiedName`*, and *`doctype`* are null, the returned DOMDocument is empty with no document element

## Errors/Exceptions

**DOM_WRONG_DOCUMENT_ERR**
: Raised if *`doctype`* has already been used with a different document or was created from a different implementation.

**DOM_NAMESPACE_ERR**
: Raised if there is an error with the namespace, as determined by *`namespaceURI`* and *`qualifiedName`*.

## See Also

- DOMDocument::__construct
- DOMImplementation::createDocumentType

# DOMImplementation::createDocumentType

DOMImplementation::createDocumentType -- Creates an empty DOMDocumentType object

## Description

DOMDocumentType **DOMImplementation::createDocumentType** ( [ string `$qualifiedName` [, string `$publicId` [, string `$systemId` ] ] ] )

Creates an empty DOMDocumentType object. Entity declarations and notations are not made available. Entity reference expansions and default attribute additions do not occur.

## Parameters

*`qualifiedName`*
The qualified name of the document type to create.

*`publicId`*
The external subset public identifier.

*`systemId`*
The external subset system identifier.

## Return Values

A new DOMDocumentType node with its *ownerDocument* set to **NULL**.

## Examples

**Example #25 - Creating a document with an attached DTD**

```
<?php

// Creates an instance of the DOMImplementation class
$imp = new DOMImplementation;

// Creates a DOMDocumentType instance
$dtd = $imp->createDocumentType('graph', '', 'graph.dtd');

// Creates a DOMDocument instance
$dom = $imp->createDocument("", "", $dtd);

// Set other properties
$dom->encoding = 'UTF-8';
$dom->standalone = false;
```

```
// Create an empty element
$element = $dom->createElement('graph');

// Append the element
$dom->appendChild($element);

// Retrieve and print the document
echo $dom->saveXML();

?>
```

The above example will output:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE graph SYSTEM "graph.dtd">
<graph/>
```

## Errors/Exceptions

**DOM_NAMESPACE_ERR**
Raised if there is an error with the namespace, as determined by *`qualifiedName`*.

## See Also

- DOMImplementation::createDocument

# DOMImplementation::hasFeature

DOMImplementation::hasFeature -- Test if the DOM implementation implements a specific feature

## Description

bool **DOMImplementation::hasFeature** ( string `$feature`, string `$version` )

Test if the DOM implementation implements a specific *`feature`*.

You can find a list of all features in the » Conformance section of the DOM specification.

## Parameters

*`feature`*
The feature to test.

*`version`*
The version number of the *`feature`* to test. In level 2, this can be either *2.0* or *1.0*.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Examples

**Example #26 - Testing your DOM Implementation**

```
<?php

$features = array(
 'Core'           => 'Core module',
 'XML'            => 'XML module',
 'HTML'           => 'HTML module',
 'Views'          => 'Views module',
 'Stylesheets'    => 'Style Sheets module',
 'CSS'            => 'CSS module',
 'CSS2'           => 'CSS2 module',
 'Events'         => 'Events module',
 'UIEvents'       => 'User interface Events module',
 'MouseEvents'    => 'Mouse Events module',
 'MutationEvents' => 'Mutation Events module',
 'HTMLEvents'     => 'HTML Events module',
 'Range'          => 'Range module',
 'Traversal'      => 'Traversal module'
);
```

```
foreach ($features as $key => $name) {
 if (DOMImplementation::hasFeature($key, '2.0')) {
   echo "Has feature $name\n";
 } else {
   echo "Missing feature $name\n";
 }
}

?>
```

## See Also

- DOMNode::isSupported

# The DOMNamedNodeMap class

## Class synopsis

**DOMNamedNodeMap**

DOMNamedNodeMap {

/* Properties */

/* Methods */

DOMNode **DOMNamedNodeMap::getNamedItem** ( string `$name` )

DOMNode **DOMNamedNodeMap::getNamedItemNS** ( string `$namespaceURI`, string `$localName` )

DOMNode **DOMNamedNodeMap::item** ( int `$index` )
}

# DOMNamedNodeMap::getNamedItem

DOMNamedNodeMap::getNamedItem -- Retrieves a node specified by name

## Description

DOMNode **DOMNamedNodeMap::getNamedItem** ( string `$name` )

Retrieves a node specified by its *nodeName*.

## Parameters

*`name`*
: The *nodeName* of the node to retrieve.

## Return Values

A node (of any type) with the specified *nodeName*, or **NULL** if no node is found.

## See Also

- DOMNamedNodeMap::getNamedItemNS

# DOMNamedNodeMap::getNamedItemNS

DOMNamedNodeMap::getNamedItemNS -- Retrieves a node specified by local name and namespace URI

## Description

DOMNode **DOMNamedNodeMap::getNamedItemNS** ( string `$namespaceURI`, string `$localName` )

Retrieves a node specified by *`localName`* and *`namespaceURI`*.

## Parameters

*`namespaceURI`*
: The namespace URI of the node to retrieve.

*`localName`*
: The local name of the node to retrieve.

## Return Values

A node (of any type) with the specified local name and namespace URI, or **NULL** if no node is found.

## See Also

- DOMNamedNodeMap::getNamedItem

# DOMNamedNodeMap::item

DOMNamedNodeMap::item -- Retrieves a node specified by index

## Description

DOMNode **DOMNamedNodeMap::item** ( int `$index` )

Retrieves a node specified by *`index`* within the DOMNamedNodeMap object.

## Parameters

*`index`*
: Index into this map.

## Return Values

The node at the *`index`* th position in the map, or **NULL** if that is not a valid index (greater than or equal to the number of nodes in this map).

# The DOMNode class

## Class synopsis

**DOMNode**

```
DOMNode {
    /*  Properties */
    public readonly string nodeName;
    public string nodeValue;
    public readonly int nodeType;
    public readonly DOMNode parentNode;
    public readonly DOMNodeList childNodes;
    public readonly DOMNode firstChild;
    public readonly DOMNode lastChild;
    public readonly DOMNode previousSibling;
    public readonly DOMNode nextSibling;
    public readonly DOMNamedNodeMap attributes;
    public readonly DOMDocument ownerDocument;
    public readonly string namespaceURI;
    public string prefix;
    public readonly string localName;
    public readonly string baseURI;
    public string textContent;
    /*  Methods */
```

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*nodeName*
Returns the most accurate name for the current node type

*nodeValue*
The value of this node, depending on its type

*nodeType*
Gets the type of the node. One of the predefined XML_xxx_NODE constants

*parentNode*
The parent of this node

*childNodes*
A DOMNodeList that contains all children of this node. If there are no children, this is an empty DOMNodeList.

*firstChild*
The first child of this node. If there is no such node, this returns **NULL**.

*lastChild*
The last child of this node. If there is no such node, this returns **NULL**.

*previousSibling*
: The node immediately preceding this node. If there is no such node, this returns **NULL**.

*nextSibling*
: The node immediately following this node. If there is no such node, this returns **NULL**.

*attributes*
: A DOMNamedNodeMap containing the attributes of this node (if it is a DOMElement ) or **NULL** otherwise.

*ownerDocument*
: The DOMDocument object associated with this node.

*namespaceURI*
: The namespace URI of this node, or **NULL** if it is unspecified.

*prefix*
: The namespace prefix of this node, or **NULL** if it is unspecified.

*localName*
: Returns the local part of the qualified name of this node.

*baseURI*
: The absolute base URI of this node or **NULL** if the implementation wasn't able to obtain an absolute URI.

*textContent*
: This attribute returns the text content of this node and its descendants.

## See Also

- » W3C specification of Node

# DOMNode::appendChild

DOMNode::appendChild -- Adds new child at the end of the children

## Description

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

This functions appends a child to an existing list of children or creates a new list of children. The child can be created with e.g. DOMDocument::createElement, DOMDocument::createTextNode etc. or simply by using any other node.

## Parameters

*newnode*
The appended child.

## Return Values

The node added.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
Raised if this node is readonly or if the previous parent of the node being inserted is readonly.

**DOM_HIERARCHY_REQUEST_ERR**
Raised if this node is of a type that does not allow children of the type of the *newnode* node, or if the node to append is one of this node's ancestors or this node itself.

**DOM_WRONG_DOCUMENT_ERR**
Raised if *newnode* was created from a different document than the one that created this node.

## Examples

The following example will add a new element node to a fresh document.

**Example #27 - Adding a child**

```
<?php

$doc = new DOMDocument;
```

```
$node = $doc->createElement("para");
$newnode = $doc->appendChild($node);

echo $doc->saveXML();
?>
```

## See Also

- DOMNode::removeChild
- DOMNode::replaceChild

# DOMNode::cloneNode

DOMNode::cloneNode -- Clones a node

## Description

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

Creates a copy of the node.

## Parameters

*deep*
: Indicates whether to copy all descendant nodes. This parameter is defaulted to **FALSE**.

## Return Values

The cloned node.

# DOMNode::hasAttributes

DOMNode::hasAttributes -- Checks if node has attributes

## Description

bool **DOMNode::hasAttributes** ( void )

This method checks if the node has attributes. The tested node have to be an **XML_ELEMENT_NODE**.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMNode::hasChildNodes

# DOMNode::hasChildNodes

DOMNode::hasChildNodes -- Checks if node has children

## Description

bool **DOMNode::hasChildNodes** ( void )

This function checks if the node has children.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMNode::hasAttributes

# DOMNode::insertBefore

DOMNode::insertBefore -- Adds a new child before a reference node

## Description

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

This function inserts a new node right before the reference node. If you plan to do further modifications on the appended child you must use the returned node.

## Parameters

*`newnode`*
The new node.

*`refnode`*
The reference node. If not supplied, *`newnode`* is appended to the children.

## Return Values

The inserted node.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
Raised if this node is readonly or if the previous parent of the node being inserted is readonly.

**DOM_HIERARCHY_REQUEST_ERR**
Raised if this node is of a type that does not allow children of the type of the *`newnode`* node, or if the node to append is one of this node's ancestors or this node itself.

**DOM_WRONG_DOCUMENT_ERR**
Raised if *`newnode`* was created from a different document than the one that created this node.

**DOM_NOT_FOUND**
Raised if *`refnode`* is not a child of this node.

# DOMNode::isDefaultNamespace

DOMNode::isDefaultNamespace -- Checks if the specified namespaceURI is the default namespace or not

## Description

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

Tells whether *`namespaceURI`* is the default namespace.

## Parameters

*`namespaceURI`*
The namespace URI to look for.

## Return Values

Return **TRUE** if *`namespaceURI`* is the default namespace, **FALSE** otherwise.

# DOMNode::isSameNode

DOMNode::isSameNode -- Indicates if two nodes are the same node

## Description

bool **DOMNode::isSameNode** ( DOMNode $node )

This function indicates if two nodes are the same node. The comparison is *not* based on content

## Parameters

*node*
The compared node.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

# DOMNode::isSupported

DOMNode::isSupported -- Checks if feature is supported for specified version

## Description

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

Checks if the asked *`feature`* is supported for the specified *`version`*.

## Parameters

*`feature`*
: The feature to test. See the example of DOMImplementation::hasFeature for a list of features.

*`version`*
: The version number of the *`feature`* to test.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- DOMImplementation::hasFeature

# DOMNode::lookupNamespaceURI

DOMNode::lookupNamespaceURI -- Gets the namespace URI of the node based on the prefix

## Description

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

Gets the namespace URI of the node based on the *`prefix`*.

## Parameters

*`prefix`*
: The prefix of the namespace.

## Return Values

The namespace URI of the node.

## See Also

- DOMNode::lookupPrefix

# DOMNode::lookupPrefix

DOMNode::lookupPrefix -- Gets the namespace prefix of the node based on the namespace URI

## Description

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

Gets the namespace prefix of the node based on the namespace URI.

## Parameters

*`namespaceURI`*
: The namespace URI.

## Return Values

The prefix of the namespace.

## See Also

- DOMNode::lookupNamespaceURI

# DOMNode::normalize

DOMNode::normalize -- Normalizes the node

## Description

void **DOMNode::normalize** ( void )

Normalizes the node.

## Return Values

No value is returned.

## See Also

- » The DOM Specification
- DOMDocument::normalizeDocument

# DOMNode::removeChild

DOMNode::removeChild -- Removes child from list of children

## Description

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

This functions removes a child from a list of children.

## Parameters

*`oldnode`*
The removed child.

## Return Values

If the child could be removed the functions returns the old child.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
Raised if this node is readonly.

**DOM_NOT_FOUND**
Raised if *`oldnode`* is not a child of this node.

## Examples

The following example will delete the chapter element of our XML document.

**Example #28 - Removing a child**

```
<?php

$doc = new DOMDocument;
$doc->load('book.xml');

$book = $doc->documentElement;

// we retrieve the chapter and remove it from the book
$chapter = $book->getElementsByTagName('chapter')->item(0);
$oldchapter = $book->removeChild($chapter);

echo $doc->saveXML();
?>
```

The above example will output:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
         "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
<book id="listing">
<title>My lists</title>

</book>
```

## See Also

- DOMNode::appendChild
- DOMNode::replaceChild

# DOMNode::replaceChild

DOMNode::replaceChild -- Replaces a child

## Description

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )

This function replaces the child *`oldnode`* with the passed new node. If the new node is already a child it will not be added a second time. If the replacement succeeds the old node is returned.

## Parameters

*`newnode`*
The new node. It must be a member of the target document, i.e. created by one of the DOMDocument->createXXX() methods or imported in the document by DOMDocument::importNode.

*`oldnode`*
The old node.

## Return Values

The old node or **FALSE** if an error occur.

## Errors/Exceptions

**DOM_NO_MODIFICATION_ALLOWED_ERR**
Raised if this node is readonly or if the previous parent of the node being inserted is readonly.

**DOM_HIERARCHY_REQUEST_ERR**
Raised if this node is of a type that does not allow children of the type of the *`newnode`* node, or if the node to put in is one of this node's ancestors or this node itself.

**DOM_WRONG_DOCUMENT_ERR**
Raised if *`newnode`* was created from a different document than the one that created this node.

**DOM_NOT_FOUND**
Raised if *`oldnode`* is not a child of this node.

## See Also

- DOMNode::appendChild
- DOMNode::removeChild

# The DOMNodeList class

## Class synopsis

**DOMNodeList**

DOMNodeList {

/* Properties */

readonly public int *length*;

/* Methods */

DOMNode **DOMNodelist::item** ( int `$index` )
}

## Properties

*length*
The number of nodes in the list. The range of valid child node indices is 0 to *length - 1* inclusive.

# DOMNodelist::item

DOMNodelist::item -- Retrieves a node specified by index

## Description

DOMNode **DOMNodelist::item** ( int `$index` )

Retrieves a node specified by *`index`* within the DOMNodeList object.

| **Tip** |
| --- |
| If you need to know the number of nodes in the collection, use the *length* property of the DOMNodeList object. |

## Parameters

*`index`*
Index of the node into the collection.

## Return Values

The node at the *`index`* th position in the DOMNodeList, or **NULL** if that is not a valid index.

## Examples

**Example #29 - Traversing all the entries of the table**

```
<?php

$doc = new DOMDocument;
$doc->load('book.xml');

$items = $doc->getElementsByTagName('entry');

for ($i = 0; $i < $items->length; $i++) {
   echo $items->item($i)->nodeValue . "\n";
}

?>
```

Alternatively, you can use foreach, which is a much more convenient way:

```
<?php
```

```
foreach ($items as $item) {
   echo $item->nodeValue . "\n";
}

?>
```

The above example will output:

```
Title
Author
Language
ISBN
The Grapes of Wrath
John Steinbeck
en
0140186409
The Pearl
John Steinbeck
en
014017737X
Samarcande
Amine Maalouf
fr
2253051209
```

# The DOMNotation class

## Class synopsis

**DOMNotation**

DOMNotation extends DOMNode {

/* Properties */

readonly public string *publicId*;

readonly public string *systemId*;

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*publicId*
: Prop description

*systemId*
: Prop description

# The DOMProcessingInstruction class

## Class synopsis

**DOMProcessingInstruction**

DOMProcessingInstruction extends DOMNode {

/* Properties */

readonly public string *target*;

public string *data*;

/* Methods */

__**construct** ( string `$name` [, string `$value` ] )

/* Inherited methods */

DOMNode **DOMNode::appendChild** ( DOMNode `$newnode` )

DOMNode **DOMNode::cloneNode** ( [ bool `$deep` ] )

bool **DOMNode::hasAttributes** ( void )

bool **DOMNode::hasChildNodes** ( void )

DOMNode **DOMNode::insertBefore** ( DOMNode `$newnode` [, DOMNode `$refnode` ] )

bool **DOMNode::isDefaultNamespace** ( string `$namespaceURI` )

bool **DOMNode::isSameNode** ( DOMNode `$node` )

bool **DOMNode::isSupported** ( string `$feature`, string `$version` )

string **DOMNode::lookupNamespaceURI** ( string `$prefix` )

string **DOMNode::lookupPrefix** ( string `$namespaceURI` )

void **DOMNode::normalize** ( void )

DOMNode **DOMNode::removeChild** ( DOMNode `$oldnode` )

DOMNode **DOMNode::replaceChild** ( DOMNode `$newnode`, DOMNode `$oldnode` )
}

## Properties

*target*
Prop description

*data*
Prop description

# DOMProcessingInstruction::__construct

DOMProcessingInstruction::__construct -- Creates a new DOMProcessingInstruction object

## Description

**DOMProcessingInstruction**

__**construct** ( string `$name` [, string `$value` ] )

Creates a new DOMProcessingInstruction object. This object is read only. It may be appended to a document, but additional nodes may not be appended to this node until the node is associated with a document. To create a writeable node, use DOMDocument::createProcessingInstruction.

## Parameters

*`name`*
The tag name of the processing instruction.

*`value`*
The value of the processing instruction.

## Examples

**Example #30 - Creating a new DOMProcessingInstruction**

```
<?php

$dom = new DOMDocument('1.0', 'UTF-8');
$html = $dom->appendChild(new DOMElement('html'));
$body = $html->appendChild(new DOMElement('body'));
$pinode = new DOMProcessingInstruction('php', 'echo "Hello World"; ');
$body->appendChild($pinode);
echo $dom->saveXML();

?>
```

The above example will output:

```
<?xml version="1.0" encoding="UTF-8"?>
<html><body><?php echo "Hello World"; ?></body></html>
```

## See Also

- DOMDocument::createProcessingInstruction

# The DOMText class

## Class synopsis

**DOMText**

DOMText extends DOMCharacterData {

/* Properties */

readonly public string *wholeText*;

/* Methods */

__**construct** ( [ string `$value` ] )

bool **DOMText::isWhitespaceInElementContent** ( void )

DOMText **DOMText::splitText** ( int `$offset` )

/* Inherited methods */

void **DOMCharacterData::appendData** ( string `$data` )

void **DOMCharacterData::deleteData** ( int `$offset`, int `$count` )

void **DOMCharacterData::insertData** ( int `$offset`, string `$data` )

void **DOMCharacterData::replaceData** ( int `$offset`, int `$count`, string `$data` )

string **DOMCharacterData::substringData** ( int `$offset`, int `$count` )
}

## Properties

*wholeText*
Prop description

# DOMText::__construct

DOMText::__construct -- Creates a new DOMText object

## Description

**DOMText**

**__construct** ( [ string `$value` ] )

Creates a new DOMText object.

## Parameters

*`value`*
The value of the text node. If not supplied an empty text node is created.

## Examples

**Example #31 - Creating a new DOMText**

```
<?php

$dom = new DOMDocument('1.0', 'iso-8859-1');
$element = $dom->appendChild(new DOMElement('root'));
$text = $element->appendChild(new DOMText('root value'));
echo $dom->saveXML(); /* <?xml version="1.0"
encoding="iso-8859-1"?><root>root value</root> */

?>
```

## See Also

- DOMDocument::createTextNode

# DOMText::isWhitespaceInElementContent

DOMText::isWhitespaceInElementContent -- Indicates whether this text node contains whitespace

## Description

bool **DOMText::isWhitespaceInElementContent** ( void )

Indicates whether this text node contains whitespace. The text node is determined to contain whitespace in element content during the load of the document.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

# DOMText::splitText

DOMText::splitText -- Breaks this node into two nodes at the specified offset

## Description

DOMText **DOMText::splitText** ( int `$offset` )

Breaks this node into two nodes at the specified *`offset`*, keeping both in the tree as siblings.

After being split, this node will contain all the content up to the *`offset`*. If the original node had a parent node, the new node is inserted as the next sibling of the original node. When the *`offset`* is equal to the length of this node, the new node has no data.

## Parameters

*`offset`*
: The offset at which to split, starting from 0.

## Return Values

The new node of the same type, which contains all the content at and after the *`offset`*.

# The DOMXPath class

## Class synopsis

**DOMXPath**

DOMXPath {

/* Properties */

public DOMDocument *document*;

/* Methods */

__**construct** ( DOMDocument `$doc` )

mixed **DOMXPath::evaluate** ( string `$expression` [, DOMNode `$contextnode` ] )

DOMNodeList **DOMXPath::query** ( string `$expression` [, DOMNode `$contextnode` ] )

bool **DOMXPath::registerNamespace** ( string `$prefix`, string `$namespaceURI` )
}

## Properties

*document*
Prop description

# DOMXPath::__construct

DOMXPath::__construct -- Creates a new DOMXPath object

## Description

**DOMXPath**

__**construct** ( DOMDocument `$doc` )

Creates a new DOMXPath object.

## Parameters

*doc*
The DOMDocument associated with the DOMXPath.

# DOMXPath::evaluate

DOMXPath::evaluate -- Evaluates the given XPath expression and returns a typed result if possible.

## Description

mixed **DOMXPath::evaluate** ( string `$expression` [, DOMNode `$contextnode` ] )

Executes the given XPath *`expression`* and returns a typed result if possible.

## Parameters

*`expression`*
: The XPath expression to execute.

*`contextnode`*
: The optional *`contextnode`* can be specified for doing relative XPath queries. By default, the queries are relative to the root element.

## Return Values

Returns a typed result if possible or a DOMNodeList containing all nodes matching the given XPath *`expression`*.

## Examples

**Example #32 - Getting the count of all the english books**

```
<?php

$doc = new DOMDocument;

$doc->load('book.xml');

$xpath = new DOMXPath($doc);

$tbody = $doc->getElementsByTagName('tbody')->item(0);

// our query is relative to the tbody node
$query = 'count(row/entry[. = "en"])';

$entries = $xpath->evaluate($query, $tbody);
echo "There are $entries english books\n";

?>
```

The above example will output:

```
There are 2 english books
```

## See Also

- DOMXPath::query

# DOMXPath::query

DOMXPath::query -- Evaluates the given XPath expression

## Description

DOMNodeList **DOMXPath::query** ( string `$expression` [, DOMNode `$contextnode` ] )

Executes the given XPath *`expression`*.

## Parameters

*`expression`*
The XPath expression to execute.

*`contextnode`*
The optional *`contextnode`* can be specified for doing relative XPath queries. By default, the queries are relative to the root element.

## Return Values

Returns a DOMNodeList containing all nodes matching the given XPath *`expression`*. Any expression which do not return nodes will return an empty DOMNodeList.

## Examples

**Example #33 - Getting all the english books**

```
<?php

$doc = new DOMDocument;

// We don't want to bother with white spaces
$doc->preserveWhiteSpace = false;

$doc->Load('book.xml');

$xpath = new DOMXPath($doc);

// We starts from the root element
$query = '//book/chapter/para/informaltable/tgroup/tbody/row/entry[. =
"en"]';

$entries = $xpath->query($query);

foreach ($entries as $entry) {
   echo "Found {$entry->previousSibling->previousSibling->nodeValue}," .
```

```
        " by {$entry->previousSibling->nodeValue}\n";
}
?>
```

The above example will output:

```
Found The Grapes of Wrath, by John Steinbeck
Found The Pearl, by John Steinbeck
```

We can also use the *contextnode* parameter to shorten our expression:

```
<?php

$doc = new DOMDocument;
$doc->preserveWhiteSpace = false;

$doc->load('book.xml');

$xpath = new DOMXPath($doc);

$tbody = $doc->getElementsByTagName('tbody')->item(0);

// our query is relative to the tbody node
$query = 'row/entry[. = "en"]';

$entries = $xpath->query($query, $tbody);

foreach ($entries as $entry) {
   echo "Found {$entry->previousSibling->previousSibling->nodeValue}," .
        " by {$entry->previousSibling->nodeValue}\n";
}
?>
```

## See Also

- DOMXPath::evaluate

# DOMXPath::registerNamespace

DOMXPath::registerNamespace -- Registers the namespace with the DOMXPath object

## Description

bool **DOMXPath::registerNamespace** ( string `$prefix`, string `$namespaceURI` )

Registers the *`namespaceURI`* and *`prefix`* with the DOMXPath object.

## Parameters

*`prefix`*
The prefix.

*`namespaceURI`*
The URI of the namespace.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

# DOM Functions

# dom_import_simplexml

dom_import_simplexml -- Gets a DOMElement object from a SimpleXMLElement object

## Description

DOMElement **dom_import_simplexml** ( SimpleXMLElement `$node` )

This function takes the node *node* of class SimpleXML and makes it into a DOMElement node. This new object can then be used as a native DOMElement node.

## Parameters

*node*
The SimpleXMLElement node.

## Return Values

The DOMElement node added or **FALSE** if any errors occur.

## Examples

**Example #34 - Import SimpleXML into DOM with dom_import_simplexml()**

```
<?php

$sxe =
simplexml_load_string('<books><book><title>blah</title></book></books>');

if ($sxe === false) {
   echo 'Error while parsing the document';
   exit;
}

$dom_sxe = dom_import_simplexml($sxe);
if (!$dom_sxe) {
   echo 'Error while converting XML';
   exit;
}

$dom = new DOMDocument('1.0');
$dom_sxe = $dom->importNode($dom_sxe, true);
$dom_sxe = $dom->appendChild($dom_sxe);

echo $dom->saveXML();

?>
```

## See Also

- simplexml_import_dom()