DOM Appendix

THE DOCUMENT OBJECT MODEL, LEVEL 1

Adapted from the ZVON (www.zvon.org) tutorial site, DOM1 reference with examples by Jiri Znamenacek, which can be found at www.zvon.org/xxl/DOM1reference/Output/index.html.
2 DOM Appendix

The following is the source document used for all the examples in this appendix (unless a source document is specifically listed within an attribute or method description).

Example: Source

```html
<div id="doc">
  <div>
    Text in the first DIV.
  </div>
  <div id="DDD" class="secondClass">
    Some text in the second DIV.
  </div>
  <div class="thirdClass">
    Some text and <span id="SSS">element</span> in the third DIV.
  </div>
  <div class="fourthClass">
    We can try <i>another element</i>. It will be much more <b>interesting</b>.
  </div>
  <div>
    Text in the last DIV.
  </div>
</div>
```

Attr

Type of interface Fundamental

Own properties

Attributes name, specified, value

Methods None.

Inherited properties

Attributes nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode
Description  The Attr interface represents an attribute in an Element object. Typically the allowable values for the attribute are defined in a document type definition.

NOTE  Although Attr objects inherit the Node interface, they are not considered part of the document tree and hence the Node attributes parentNode, previousSibling, and nextSibling have a null value for them.

NOTE  In XML, where the value of an attribute can contain entity references, the child nodes of the Attr node provide a representation in which entity references are not expanded. These child nodes may be either Text or EntityReference nodes. Because the attribute type may be unknown, there are no tokenized attribute values.

attribute: Attr.name

Read-only  Yes
Type  DOMString
Description  Returns the name of this attribute.

Example
var attr = document.createAttribute('temp');
var output = attr.name;

attribute: Attr.specified

Read-only  Yes
Type  Boolean
Description  Thanks to this attribute, you can find out something about the DTD background of a tested attribute. In summary:

- If the attribute has an assigned value in the document, then specified is true, and the value is the assigned value.
4 DOM Appendix

- If the attribute has no assigned value in the document and has a default value in the DTD, then specified is false, and the value is the default value in the DTD.

- If the attribute has no assigned value in the document and has a value of #IMPLIED in the DTD, then the attribute does not appear in the structure model of the document.

Example

```javascript
var attr = document.createAttribute('temp');
attr.value = 'temporary';
var output = attr.specified;
```

attribute: Attr.value

- Read-only: No
- Type: DOMString
- Description: On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. On setting, this creates a Text node with the unparsed contents of the string.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

Example

```javascript
var attr = document.createAttribute('temp');
attr.value = 'temporary';
var output = attr.value;
```

attribute: Attr.nodeName (inherited from Node)

- Read-only: Yes
- Type: DOMString
Description  The name of this node, depending on its type.

Example

<!-- See example under Document interface -->

**attribute: Attr.nodeValue (inherited from Node)**

Read-only  No
Type  DOMString
Description  The value of this node, depending on its type.

**Exceptions**

**DOMException NO_MODIFICATION_ALLOWED_ERR**
This exception raises on setting when the node is read-only.

**DOMException DOMSTRING_SIZE_ERR**  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

<!-- See example under Document interface -->

**attribute: Attr.nodeType (inherited from Node)**

Read-only  Yes
Type  unsigned short
Description  A code representing the type of the underlying object.

Example

<!-- See example under Document interface -->
attribute: **Attr.parentNode** (inherited from Node)

Read-only  Yes

Type  Node

**Description**  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

**Example**

<!-- See example under Document interface -->

attribute: **Attr.childNodes** (inherited from Node)

Read-only  Yes

Type  NodeList

**Description**  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

**NOTE**

The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

**Example**

<!-- See example under Document interface -->

attribute: **Attr.firstChild** (inherited from Node)

Read-only  Yes
Type     Node

**Description**  The first child of this node. If there is no such node, this returns null.

---

**Example**

<!-- See example under Document interface -->

**attribute: Attr.firstChild (inherited from Node)**

**Read-only**  Yes

**Type**     Node

**Description**  The last child of this node. If there is no such node, this returns null.

---

**Example**

```javascript
var main = document.getElementById('doc');
var output = main.lastChild.nodeName;
```

**attribute: Attr.previousSibling (inherited from Node)**

**Read-only**  Yes

**Type**     Node

**Description**  The node immediately preceding this node. If there is no such node, this returns null.

---

**Example**

<!-- See example under Document interface -->

**attribute: Attr.nextSibling (inherited from Node)**

**Read-only**  Yes

**Type**     Node
8  DOM Appendix

**Description**  The node immediately following this node. If there is no such node, this returns null.

**Example**

<!-- See example under Document interface -->

attribute: `Attr.attributes` (inherited from Node)

**Read-only**  Yes

**Type**  NamedNodeMap

**Description**  A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

**Example**

<!-- See example under Document interface -->

attribute: `Attr.ownerDocument` (inherited from Node)

**Read-only**  Yes

**Type**  Document

**Description**  The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

**Example**

<!-- See example under Document interface -->

method: `Attr.insertBefore(newChild, refChild)` (inherited from Node)

**Description**  Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.
Parameters  Node newChild—The node to insert. Node refChild—The reference node, i.e., the node before which the new node must be inserted.

Returns  Node—The node being inserted.

Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if refChild is not a child of this node.

NOTE
If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If the newChild is already in the tree, it is first removed.

Example

<!-- See example under Document interface -->

method: Attr.replaceChild(newChild, oldChild) (inherited from Node)

Description  Replaces the child node oldChild with newChild in the list of children and returns the oldChild node. If the newChild is already in the tree, it is first removed.

Parameters  Node newChild—The new node to put in the child list. Node oldChild—The node being replaced in the list.

Returns  Node—The node replaced.
Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

<!-- See example under Document interface -->

Example

<!-- See example under Document interface -->

method: Attr.removeChild(oldChild)
(inherited from Node)

Description  Removes the child node indicated by oldChild from the list of children and returns it.

Parameters  Node oldChild—The node being removed.

Returns  Node—The node removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

<!-- See example under Document interface -->
method: Attr.appendChild(newChild) (inherited from Node)

Description   Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.

Parameters   Node newChild—The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

Returns   Node—The node added.

Exceptions

DOMException HIERARCHY_REQUEST_ERR   Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR   Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR   Raised if this node is read-only.

Example

<!-- See example under Document interface -->

method: Attr.hasChildNodes() (inherited from Node)

Description   This is a convenient method to allow easy determination of whether a node has any children.

Parameters   None.

Returns   Boolean—True if the node has any children, false if the node has no children.

Exceptions   None.
method: Attr.cloneNode(deep) (inherited from Node)

Description  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

Parameters  Boolean deep—If true, recursively clone the subtree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

Returns  Node—The duplicate node.

Exceptions  None.

NOTE
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example
<!-- See example under Document interface -->

CDATASection

Type of interface  Extended

Own properties

Attributes  None.

Methods  None.
Inherited properties

Attributes  data, length, nodeName, nodeValue, 
nodeType, parentNode, childNodes, firstChild, 
lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods  substringData, appendData, insertData, 
deleteData, replaceData, insertBefore, replaceChild, 
removeChild, appendChild, hasChildNodes, cloneNode, 
splitText

Description  Every CDATA-section in an XML document transforms into the node of the type CDATASection in the DOM. The CDATASection interface inherits the CharacterData interface through the Text interface. Adjacent CDATASections nodes are not merged by use of the Element.normalize() method.

attribute: CDATASection.data (inherited from CharacterData)

Read-only  No
Type  DOMString
Description  The character data of the node that implements this interface.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
var output = txt.data;
```
attribute: `CDATASEction.length` (inherited from CharacterData)

Read-only  Yes
Type      unsigned long
Description  The number of 16-bit units that are available through data and the substringData() method.

Example
var txt = document.createTextNode('AAA-BBB');
var output = txt.length;

attribute: `CDATASEction.nodeName` (inherited from Node)

Read-only  Yes
Type      DOMString
Description  The name of this node, depending on its type.

Example
var main = document.getElementById('doc');
var output1 = main.nodeName;
var output2 = document.getElementById('SSS').nodeName;

attribute: `CDATASEction.nodeValue` (inherited from Node)

Read-only  No
Type      DOMString
Description  The value of this node, depending on its type.

Exceptions
DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.
DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

```javascript
var output =
document.getElementById('DDD').firstChild.nodeValue;
```

**attribute: CDATASection.nodeType**  
(inherited from Node)

- **Read-only**  Yes
- **Type**  unsigned short
- **Description**  A code representing the type of the underlying object.

Example

```javascript
var elem = document.getElementById('DDD');
var output1 = elem.nodeType;
var output2 = elem.firstChild.nodeType;
```

**attribute: CDATASection.parentNode**  
(inherited from Node)

- **Read-only**  Yes
- **Type**  Node
- **Description**  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example

```javascript
var elem = document.getElementById('SSS');
var output = elem.parentNode.nodeName;
```
attribute: `CDATASection.childNodes`  
(inherited from Node)

Read-only  Yes
Type           NodeList
Description  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

NOTE
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the `getElementsByTagName()` method.

Example
```javascript
var main = document.getElementById('doc');
var output1 = main.childNodes.length;
var output2 = main.childNodes[0].nodeType;
```

attribute: `CDATASection.firstChild`  
(inherited from Node)

Read-only  Yes
Type           Node
Description  The first child of this node. If there is no such node, this returns null.

Example
```javascript
var main = document.getElementById('doc');
var output = main.firstChild.nodeName;
```
attribute: CDATASEction.lastChild (inherited from Node)

Read-only  Yes
Type  Node
Description  The last child of this node. If there is no such node, this returns null.

Example
var main = document.getElementById('doc');
var output = main.lastChild.nodeName;

attribute: CDATASEction.previousSibling (inherited from Node)

Read-only  Yes
Type  Node
Description  The node immediately preceding this node. If there is no such node, this returns null.

Example
var elem = document.getElementById('SSS');
var output = elem.previousSibling.nodeValue;

attribute: CDATASEction.nextSibling (inherited from Node)

Read-only  Yes
Type  Node
Description  The node immediately following this node. If there is no such node, this returns null.

Example
var elem = document.getElementById('SSS');
var output = elem.nextSibling.nodeValue;
attribute: `CDATASEction.attributes` (inherited from Node)

Read-only   Yes
Type        NamedNodeMap
Description  A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Example

```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var output = attrNode.length;
```

attribute: `CDATASEction.ownerDocument` (inherited from Node)

Read-only   Yes
Type        Document
Description  The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example

```javascript
var main = document.getElementById('doc');
var output = main.ownerDocument.documentElement.nodeName;
```

method: `CDATASEction.substringData(offset, count)` (inherited from CharacterData)

Description  Extracts a range of data from the node.
Parameters    unsigned long offset—Start offset of substring to extract. unsigned long count—The number of 16-bit units to extract.
Type          DOMString
Returns 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.

Exceptions

DOMException INDEX_SIZE_ERR Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMException DOMSTRING_SIZE_ERR Raised if the specified range of text does not fit into a DOMString.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
var output = txt.substringData(2,3);
```

method: CDATASection.appendData(arg) (inherited from CharacterData)

Description Appends the string to the end of the character data of the node.

Parameters DOMString arg—The DOMString to append.

Returns Nothing.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
txt.appendData('CCC');
var output = txt.data;
```

method: CDATASection.insertData(offset, arg) (inherited from CharacterData)

Description Inserts a string at the specified 16-bit units offset.
Parameters  unsigned long offset—The 16-bit units offset at which to insert. DOMString arg—The DOMString to insert.

Returns  Nothing.

Exceptions

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of 16-bit units in data.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
txt.insertData(3, 'CCC');
var output = txt.data;
```

method: CDATASection.deleteData(offset, count) (inherited from CharacterData)

Description  Removes a range of 16-bit units from the node. Upon success, data and length reflect the change.

Parameters  unsigned long offset—The offset from which to remove characters. unsigned long count—The number of 16-bit units to delete. If the sum of offset and count exceeds length, then all 16-bit units from offset to the end of the data are deleted.

Returns  Nothing.

Exceptions

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of characters in data, or if the specified count is negative.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.
Example

```javascript
var txt = document.createTextNode('AAA-BBB');
    txt.deleteData(0, 4);
    var output = txt.data;
```

**method: CDATASection.replaceData**

(offset, count, arg) (inherited from CharacterData)

**Description**  Replaces the characters starting at the specified 16-bit units offset with the specified string.

**Parameters**  
- unsigned long offset—The offset from which to start replacing.
- unsigned long count—The number of 16-bit units to replace. If the sum of offset and count exceeds length, then all 16-bit units to the end of the data are replaced.
- DOMString arg—The DOMString with which the range must be replaced.

**Returns**  Nothing.

**Exceptions**

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
    txt.replaceData(0, 4, 'DDDD');
    var output = txt.data;
```

**method: CDATASection.insertBefore**

(newChild, refChild) (inherited from Node)

**Description**  Inserts the node newChild before the existing child node refChild. If refChild is null, inserts newChild at the end of the list of children.
Parameters

Node `newChild`—The node to insert. Node `refChild`—The reference node, i.e., the node before which the new node must be inserted.

Returns

Node—The node being inserted.

Exceptions

`DOMException HIERARCHY_REQUEST_ERR` Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors.

`DOMException WRONG_DOCUMENT_ERR` Raised if `newChild` was created from a different document than the one that created this node.

`DOMException NO_MODIFICATION_ALLOWED_ERR` Raised if this node is read-only.

`DOMException NOT_FOUND_ERR` Raised if `refChild` is not a child of this node.

NOTE

If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

Example

```javascript
var main = document.getElementById('doc');
main.insertBefore(document.getElementById('DDD'), main.firstChild);
var output = main.firstChild.firstChild.nodeValue;
```

**method: CDATASection.replaceChild**

*(newChild, oldChild) (inherited from Node)*

Description

Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed.
Parameters
Node newChild—The new node to put in the child list. Node oldChild—The node being replaced in the list.

Returns
Node—The node replaced.

Exceptions
DOMException HIERARCHY_REQUEST_ERR Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

DOMException NOT_FOUND_ERR Raised if oldChild is not a child of this node.

Example
var elem = document.getElementById('DDD');

    elem.removeChild(document.getElementById('SSS').firstChild,
    elem.firstChild);
    var output = elem.firstChild.nodeValue;

method: CDATASection.removeChild (oldChild) (inherited from Node)

Description
Removes the child node indicated by oldChild from the list of children and returns it.

Parameters
Node oldChild—The node being removed.

Returns
Node—The node removed.

Exceptions
DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.
DOM Exception NOT_FOUND_ERR Raised if oldChild is not a child of this node.

Example

```javascript
var elem = document.getElementById('SSS');
var output1 = elem.firstChild.nodeValue;
elem.removeChild(elem.firstChild);
var output2 = elem.firstChild;
```

**method: CDATASection.appendChild (newChild) (inherited from Node)**

- **Description** Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.
- **Parameters** Node newChild—The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.
- **Returns** Node—The node added.

**Exceptions**

- **DOMException HIERARCHY_REQUEST_ERR** Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node’s ancestors.
- **DOMException WRONG_DOCUMENT_ERR** Raised if newChild was created from a different document than the one that created this node.
- **DOMException NO_MODIFICATION_ALLOWED_ERR** Raised if this node is read-only.

Example

```javascript
var main = document.getElementById('doc');
main.appendChild(main.childNodes[1]);
var output = main.lastChild.firstChild.nodeValue;
```
### CDATASection.hasChildNodes()

**Description**
This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**
None.

**Returns**
Boolean—True if the node has any children, false if the node has no children.

**Exceptions**
None.

#### Example
```javascript
var main = document.getElementById('doc');
var output1 = main.hasChildNodes();
var output2 = main.firstChild.hasChildNodes();
```

### CDATASection.cloneNode(deep)

**Description**
Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns null).

**Parameters**
- Boolean `deep`—If true, recursively clones the subtree under the specified node; if false, clones only the node itself (and its attributes, if it is an Element).

**Returns**
Node—The duplicate node.

**Exceptions**
None.

**NOTE**
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.
Example

```javascript
var main = document.getElementById('doc');
var elem = main.childNodes[3];
var returns = elem.cloneNode(true);
var output = returns.firstChild.nodeValue;
```

**method: `CDATASection.splitText(offset)` (inherited from `Text`)**

**Description**
Breaks this Text node into two Text nodes at the specified offset, keeping both in the tree as siblings. This node then only contains all the content up to the offset point. A new Text node, which is inserted as the next sibling of this node, contains all the content at and after the offset point.

**Parameters**
- `offset`—The 16-bit unit offset at which to split, starting from 0.

**Returns**
- `Text`—The new Text node.

**Exceptions**
- `DOMException INDEX_SIZE_ERR` Raised if the specified offset is negative or greater than the number of 16-bit unit in data.
- `DOMException NO_MODIFICATION_ALLOWED_ERR` Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB-CCC');
var output1 = txt.splitText(4).data;
var output2 = txt.data;
```

**CHARACTERDATA**

**Type of interface**
Fundamental

**Own properties**

**Attributes**
- `data`, `length`
Methods  substringData, appendData, insertData, deleteData, replaceData

Inherited properties

Attributes  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods  insertBefore, replaceChild, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

Description  The CharacterData interface extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to CharacterData, though Text and others do inherit the interface from it. All offsets in this interface start from 0.

attribute: CharacterData.data

Read-only  No
Type  DOMString
Description  The character data of the node that implements this interface.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

var txt = document.createTextNode('AAA-BBB');
var output = txt.data;
attribute: `CharacterData.length`

- **Read-only:** Yes
- **Type:** unsigned long
- **Description:** The number of 16-bit units that are available through data and the `substringData()` method.

**Example**

```javascript
var txt = document.createTextNode('AAA-BBB');
var output = txt.length;
```

attribute: `CharacterData.nodeName`
*(inherited from Node)*

- **Read-only:** Yes
- **Type:** DOMString
- **Description:** The name of this node, depending on its type.

**Example**

```javascript
var main = document.getElementById('doc');
var output1 = main.nodeName;
var output2 = document.getElementById('SSS').nodeName;
```

attribute: `CharacterData.nodeValue`
*(inherited from Node)*

- **Read-only:** No
- **Type:** DOMString
- **Description:** The value of this node, depending on its type.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.
DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example
var output =
document.getElementById('DDD').firstChild.nodeValue;

attribute: CharacterData.nodeType
(inherited from Node)

Read-only  Yes
Type  unsigned short
Description  A code representing the type of the underlying object.

Example
var elem = document.getElementById('DDD');
var output1 = elem.nodeType;
var output2 = elem.firstChild.nodeType;

attribute: CharacterData.parentNode
(inherited from Node)

Read-only  Yes
Type  Node
Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example
var elem = document.getElementById('SSS');
var output = elem.parentNode.nodeName;
attribute: `CharacterData.childNodes` (inherited from Node)

- **Read-only**: Yes
- **Type**: NodeList
- **Description**: A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

**NOTE**
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the `getElementsByTagName()` method.

**Example**
```javascript
var main = document.getElementById('doc');
var output1 = main.childNodes.length;
var output2 = main.childNodes[0].nodeType;
```

attribute: `CharacterData.firstChild` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The first child of this node. If there is no such node, this returns null.

**Example**
```javascript
var main = document.getElementById('doc');
var output = main.firstChild.nodeName;
```
attribute: `CharacterData.lastChild` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The last child of this node. If there is no such node, this returns null.

```
var main = document.getElementById('doc');
var output = main.lastChild.nodeName;
```

attribute: `CharacterData.previousSibling` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately preceding this node. If there is no such node, this returns null.

```
var elem = document.getElementById('SSS');
var output = elem.previousSibling.nodeValue;
```

attribute: `CharacterData.nextSibling` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately following this node. If there is no such node, this returns null.

```
var elem = document.getElementById('SSS');
var output = elem.nextSibling.nodeValue;
```
attribute: `CharacterData.attributes`  
(inherited from Node)

- **Read-only**: Yes
- **Type**: NamedNodeMap
- **Description**: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

**Example**

```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var output = attrNode.length;
```

attribute: `CharacterData.ownerDocument`  
(inherited from Node)

- **Read-only**: Yes
- **Type**: Document
- **Description**: The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

**Example**

```javascript
var main = document.getElementById('doc');
var output = main.ownerDocument.documentElement.nodeName;
```

method: `CharacterData.substringData(offset, count)`  
(offset, count)

- **Description**: Extracts a range of data from the node.
- **Parameters**
  - unsigned long `offset`—Start offset of substring to extract.
  - unsigned long `count`—The number of 16-bit units to extract.
- **Type**: DOMString
The Document Object Model, Level 1 33

Returns  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.

Exceptions

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMException DOMSTRING_SIZE_ERR  Raised if the specified range of text does not fit into a DOMString.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
var output = txt.substringData(2,3);
```

method: CharacterData.appendData(arg)

Description  Appends the string to the end of the character data of the node.

Parameters  DOMString arg—The DOMString to append.

Returns  Nothing.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
txt.appendData('CCC');
var output = txt.data;
```

method: CharacterData.insertData(offset, arg)

Description  Inserts a string at the specified 16-bit units offset.

Parameters  offset—The offset at which to insert the string.

Returns  Nothing.

Exceptions

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMException DOMSTRING_SIZE_ERR  Raised if the specified range of text does not fit into a DOMString.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
var output = txt.insertData(2, 'CCC');
var output = txt.data;
```
Parameters

- unsigned long offset—The 16-bit units offset at which to insert.
- DOMString arg—The DOMString to insert.

Returns

Nothing.

Exceptions

- DOMException INDEX_SIZE_ERR Raised if the specified offset is negative or greater than the number of 16-bit units in data.
- DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
txt.insertData(3, 'CCC');
var output = txt.data;
```

**method: CharacterData.deleteData**

**Parameters**

- unsigned long offset—The offset from which to remove characters.
- unsigned long count—The number of 16-bit units to delete. If the sum of offset and count exceeds length, then all 16-bit units from offset to the end of the data are deleted.

Returns

Nothing.

Exceptions

- DOMException INDEX_SIZE_ERR Raised if the specified offset is negative or greater than the number of characters in data, or if the specified count is negative.
- DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.
Example
var txt = document.createTextNode('AAA-BBB');
txt.deleteData(0, 4);
var output = txt.data;

**method: CharacterData.replaceData** (offset, count, arg)

**Description** Replaces the characters starting at the specified 16-bit units offset with the specified string.

**Parameters**
- unsigned long offset—The offset from which to start replacing.
- unsigned long count—The number of 16-bit units to replace. If the sum of offset and count exceeds length, then all 16-bit units to the end of the data are replaced.
- DOMString arg—The DOMString with which the range must be replaced.

**Returns** Nothing.

**Exceptions**

DOMException INDEX_SIZE_ERR Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

Example
var txt = document.createTextNode('AAA-BBB');
txt.replaceData(0, 4, 'DDDD');
var output = txt.data;

**method: CharacterData.insertBefore** (newChild, refChild) (inherited from Node)

**Description** Inserts the node newChild before the existing child node refChild. If refChild is null, inserts newChild at the end of the list of children.
Parameters

Node newChild—The node to insert. Node refChild—The reference node, i.e., the node before which the new node must be inserted.

Returns

Node—The node being inserted.

Exceptions

DOMException HIERARCHY_REQUEST_ERR Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

DOMException NOT_FOUND_ERR Raised if refChild is not a child of this node.

NOTE

If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If the newChild is already in the tree, it is first removed.

Example

```javascript
var main = document.getElementById('doc');
main.insertBefore(document.getElementById('DDD'), main.firstChild);
var output = main.firstChild.firstChild.nodeValue;
```

**method: CharacterData.replaceChild (newChild, oldChild) (inherited from Node)**

Description

Replaces the child node oldChild with newChild in the list of children and returns the oldChild node. If the newChild is already in the tree, it is first removed.
**Parameters**  
Node `newChild`—The new node to put in the child list.  
Node `oldChild`—The node being replaced in the list.

**Returns**  
Node—The node replaced.

**Exceptions**

**DOMException HIERARCHY_REQUEST_ERR**  
Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR**  
Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**  
Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  
Raised if `oldChild` is not a child of this node.

**Example**

```javascript
var elem = document.getElementById('DDD');

elem.replaceChild(document.getElementById('SSS').firstChild, elem.firstChild);
var output = elem.firstChild.nodeValue;
```

**method: CharacterData.removeChild (oldChild)**  
*(inherited from Node)*

**Description**  
Removes the child node indicated by `oldChild` from the list of children and returns it.

**Parameters**  
Node `oldChild`—The node being removed.

**Returns**  
Node—The node removed.

**Exceptions**

**DOMException NO_MODIFICATION_ALLOWED_ERR**  
Raised if this node is read-only.
DOM Exception NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example
var elem = document.getElementById('SSS');
var output1 = elem.firstChild.nodeValue;
elem.removeChild(elem.firstChild);
var output2 = elem.firstChild;

method: CharacterData.appendChild (newChild) (inherited from Node)

Description  Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.

Parameters  Node newChild—The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

Returns  Node—The node added.

Exceptions
DOM Exception HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors.

DOM Exception WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOM Exception NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example
var main = document.getElementById('doc');
main.appendChild(main.childNodes[1]);
var output = main.lastChild.firstChild.nodeValue;
method: CharacterData.hasChildNodes()
(inherited from Node)

**Description**  This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  None.

**Returns**  Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  None.

**Example**

```javascript
var main = document.getElementById('doc');
var output1 = main.hasChildNodes();
var output2 = main.firstChild.hasChildNodes();
```

method: CharacterData.cloneNode(deep)
(inherited from Node)

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

**Parameters**  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

**NOTE**

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.
Example

```javascript
var main = document.getElementById('doc');
var elem = main.childNodes[3];
var returns = elem.cloneNode(true);
var output = returns.firstChild.nodeValue;
```

**COMMENT**

**Type of interface**  Fundamental

**Own properties**

- **Attributes**  None.
- **Methods**  None.

**Inherited properties**

- **Attributes**  `data`, `length`, `nodeName`, `nodeValue`, `nodeType`, `parentNode`, `childNodes`, `firstChild`, `lastChild`, `previousSibling`, `nextSibling`, `attributes`, `ownerDocument`
- **Methods**  `substringData`, `appendData`, `insertData`, `deleteData`, `replaceData`, `insertBefore`, `replaceChild`, `removeChild`, `appendChild`, `hasChildNodes`, `cloneNode`

**Description**  Comments in XML and HTML (i.e., all characters between `<!--` and `-->`) are accessible from the Comment object in DOM.

**attribute: Comment.data (inherited from CharacterData)**

- **Read-only**  No
- **Type**  `DOMString`
- **Description**  The character data of the node that implements this interface.
Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example
var txt = document.createTextNode('AAA-BBB');
var output = txt.data;

attribute: Comment.length (inherited from CharacterData)

Read-only Yes
Type unsigned long
Description The number of 16-bit units that are available through data and the substringData() method.

Example
var txt = document.createTextNode('AAA-BBB');
var output = txt.length;

attribute: Comment.nodeName (inherited from Node)

Read-only Yes
Type DOMString
Description The name of this node, depending on its type.

Example
var main = document.getElementById('doc');
var output1 = main.nodeName;
var output2 = document.getElementById('SSS').nodeName;
attribute: Comment.nodeValue (inherited from Node)

- Read-only: No
- Type: DOMString
- Description: The value of this node, depending on its type.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR
This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

```javascript
var output =
document.getElementById('DDD').firstChild.nodeValue;
```

attribute: Comment.nodeType (inherited from Node)

- Read-only: Yes
- Type: unsigned short
- Description: A code representing the type of the underlying object.

Example

```javascript
var elem = document.getElementById('DDD');
var output1 = elem.nodeType;
var output2 = elem.firstChild.nodeType;
```

attribute: Comment.parentNode (inherited from Node)

- Read-only: Yes
- Type: Node
**Description**  The parent of this node. All nodes, except `Document`, `DocumentFragment`, and `Attr`, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

**Example**
```javascript
var elem = document.getElementById('SSS');
var output = elem.parentNode.nodeName;
```

**attribute: `Comment.childNodes` (inherited from Node)**

- **Read-only**  Yes
- **Type**  `NodeList`

**Description**  A `NodeList` that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes.

**NOTE**
The content of the returned `NodeList` is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the `NodeList` accessors; it is not a static snapshot of the content of the node. This is true for every `NodeList`, including the ones returned by the `getElementsByTagName()` method.

**Example**
```javascript
var main = document.getElementById('doc');
var output1 = main.childNodes.length;
var output2 = main.childNodes[0].nodeType;
```

**attribute: `Comment.firstChild` (inherited from Node)**

- **Read-only**  Yes
- **Type**  `Node`

**Description**  The first child of this node. If there is no such node, this returns null.
Example

```javascript
var main = document.getElementById('doc');
var output = main.firstChild.nodeName;
```

attribute: `Comment.lastChild` (inherited from Node)

Read-only: Yes
Type: Node
Description: The last child of this node. If there is no such node, this returns null.

Example

```javascript
var main = document.getElementById('doc');
var output = main.lastChild.nodeName;
```

attribute: `Comment.previousSibling` (inherited from Node)

Read-only: Yes
Type: Node
Description: The node immediately preceding this node. If there is no such node, this returns null.

Example

```javascript
var elem = document.getElementById('SSS');
var output = elem.previousSibling.nodeValue;
```

attribute: `Comment.nextSibling` (inherited from Node)

Read-only: Yes
Type: Node
Description: The node immediately following this node. If there is no such node, this returns null.
Example
var elem = document.getElementById('SSS');
var output = elem.nextSibling.nodeValue;

attribute: `Comment.attributes` (inherited from Node)

Read-only  Yes
Type  NamedNodeMap
Description  A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Example
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var output = attrNode.length;

attribute: `Comment.ownerDocument` (inherited from Node)

Read-only  Yes
Type  Document
Description  The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example
var main = document.getElementById('doc');
var output = main.ownerDocument.documentElement.nodeName;

method: `Comment.substringData(offset, count)` (inherited from CharacterData)

Description  Extracts a range of data from the node.
Parameters  unsigned long offset—Start offset of substring to extract. unsigned long count—The number of 16-bit units to extract.
Returns DOMString—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.

Exceptions

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.

DOMException DOMSTRING_SIZE_ERR  Raised if the specified range of text does not fit into a DOMString.

Example

var txt = document.createTextNode('AAA-BBB');
var output = txt.substringData(2,3);

method: Comment.appendData(arg) (inherited from CharacterData)

Description  Appends the string to the end of the character data of the node.

Parameters  DOMString arg—The DOMString to append.

Returns  Nothing.

Exception

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

var txt = document.createTextNode('AAA-BBB');
txt.appendData('CCC');
var output = txt.data;

method: Comment.insertData(offset, arg) (inherited from CharacterData)

Description  Inserts a string at the specified 16-bit units offset.
**Parameters**
unsigned long offset The 16-bit units offset at which to insert. DOMString arg—The DOMString to insert.

**Returns** Nothing.

**Exceptions**
- **DOMException INDEX_SIZE_ERR** Raised if the specified offset is negative or greater than the number of 16-bit units in data.
- **DOMException NO_MODIFICATION_ALLOWED_ERR** Raised if this node is read-only.

**Example**
```
var txt = document.createTextNode('AAA-BBB');
txt.insertData(3, 'CCC');
var output = txt.data;
```

**method: Comment.deleteData(offset, count) (inherited from CharacterData)**

**Description** Removes a range of 16-bit units from the node. Upon success, data and length reflect the change.

**Parameters**
- unsigned long offset—The offset from which to remove characters.
- unsigned long count—The number of 16-bit units to delete. If the sum of offset and count exceeds length, then all 16-bit units from offset to the end of the data are deleted.

**Returns** Nothing.

**Exceptions**
- **DOMException INDEX_SIZE_ERR** Raised if the specified offset is negative or greater than the number of characters in data, or if the specified count is negative.
- **DOMException NO_MODIFICATION_ALLOWED_ERR** Raised if this node is read-only.
method: `Comment.replaceData(offset, count, arg)` (inherited from CharacterData)

Description  Replaces the characters starting at the specified 16-bit units offset with the specified string.

Parameters  
- `offset`—The offset from which to start replacing.
- `count`—The number of 16-bit units to replace. If the sum of `offset` and `count` exceeds `length`, then all 16-bit units to the end of the data are replaced.
- `arg`—The `DOMString` with which the range must be replaced.

Returns  Nothing.

Exceptions

- **DOMException INDEX_SIZE_ERR**  Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.
- **DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

Example

```javascript
var txt = document.createTextNode('AAA-BBB');
txt.replaceData(0, 4, 'DDDD');
var output = txt.data;
```

method: `Comment.insertBefore(newChild, refChild)` (inherited from Node)

Description  Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.

Parameters  
- `newChild`—The node to insert. Node `refChild`—The reference node, i.e., the node before which the new node must be inserted.

Returns  Node—The node being inserted.
Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if refChild is not a child of this node.

NOTE

If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If the newChild is already in the tree, it is first removed.

Example

```javascript
var main = document.getElementById('doc');
main.insertBefore(document.getElementById('DDD'), main.firstChild);
var output = main.firstChild.firstChild.nodeValue;
```

method: Comment.replaceChild(newChild, oldChild) (inherited from Node)

Description  Replaces the child node oldChild with newChild in the list of children and returns the oldChild node. If the newChild is already in the tree, it is first removed.

Parameters  Node newChild–The new node to put in the child list. Node oldChild–The node being replaced in the list.

Returns  Node–The node replaced.
Exceptions

DOMException HIERARCHY_REQUEST_ERR Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node's ancestors.

DOMException WRONG_DOCUMENT_ERR Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

DOMException NOT_FOUND_ERR Raised if oldChild is not a child of this node.

Example

```javascript
var elem = document.getElementById('DDD');

elem.replaceChild(document.getElementById('SSS').firstChild,
                 elem.firstChild);
var output = elem.firstChild.nodeValue;
```

method: `Comment.removeChild(oldChild)` (inherited from `Node`)

Description Removes the child node indicated by oldChild from the list of children, and returns it.

Parameters Node oldChild—The node being removed.

Returns Node—The node removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

DOMException NOT_FOUND_ERR Raised if oldChild is not a child of this node.
Example

```javascript
var elem = document.getElementById('SSS');
var output1 = elem.firstChild.nodeValue;
elem.removeChild(elem.firstChild);
var output2 = elem.firstChild;
```

**method: `Comment.appendChild(newChild)`**

*(inherited from Node)*

**Description**
Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

**Parameters**
- `Node newChild`—The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

**Returns**
- `Node`—The node added.

**Exceptions**

- **DOMException HIERARCHY_REQUEST_ERR**
  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.

- **DOMException WRONG_DOCUMENT_ERR**
  Raised if `newChild` was created from a different document than the one that created this node.

- **DOMException NO_MODIFICATION_ALLOWED_ERR**
  Raised if this node is read-only.

**Example**

```javascript
var main = document.getElementById('doc');
main.appendChild(main.childNodes[1]);
var output = main.lastChild.firstChild.nodeValue;
```
**method: Comment.hasChildNodes()**
(inherited from Node)

**Description**  This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  None.

**Returns**  Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  None.

**Example**

```javascript
var main = document.getElementById('doc');
var output1 = main.hasChildNodes();
var output2 = main.firstChild.hasChildNodes();
```

**method: Comment.cloneNode(deep)**
(inherited from Node)

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

**Parameters**  Boolean deep—If true, recursively clone the subtree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

**NOTE**

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.
Example

```javascript
var main = document.getElementById('doc');
var elem = main.childNodes[3];
var returns = elem.cloneNode(true);
var output = returns.firstChild.nodeValue;
```

**Document**

**Type of interface** Fundamental

**Own properties**

**Attributes**
doctype, implementation, documentElement

**Methods**
createElement, createDocumentFragment, createTextNode, createComment, createCDATASection, createProcessingInstruction, createAttribute, createEntityReference, getElementsByTagName

**Inherited properties**

**Attributes**
nodename, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

**Methods**
insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

**Description**
The Document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree and provides the primary access to the document's data. Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node objects created have an ownerDocument attribute that associates them with the Document within whose context they were created.

**Example: Source**

```html
Example: Source
<br>
<div id="doc">
  <div>
    Text in the first DIV.
  </div>
</div>
```
54  DOM Appendix

attribute: Document.doctype

Read-only  Yes
Type  DocumentType
Description  Stores the document type declaration associated with document (or null if the DTD was not specified).

Example
var output = document.doctype;

attribute: Document.implementation

Read-only  Yes
Type  DOMImplementation
Description  The DOMImplementation object that handles this document.

Example
var output = document.implementation.hasFeature('HTML', '1.0');
attribute: Document.documentElement

   Read-only   Yes
   Type        Element
   Description Stores the root element of the document.

Example
   var output = document.documentElement.nodeName;

attribute: Document.nodeName (inherited from Node)

   Read-only   Yes
   Type        DOMString
   Description The name of this node, depending on its type.

Example
   var main = document.getElementById('doc');
   var output1 = main.nodeName;
   var output2 = document.getElementById('SSS').nodeName;

attribute: Document.nodeValue (inherited from Node)

   Read-only   No
   Type        DOMString
   Description The value of this node, depending on its type.

Exceptions

   DOMException NO_MODIFICATION_ALLOWED_ERR
   This exception raises on setting when the node is read-only.

   DOMException DOMSTRING_SIZE_ERR   This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.
Example
var output =
document.getElementById('DDD').firstChild.nodeValue;

attribute: Document.nodeType (inherited from Node)

| Read-only | Yes |
| Type      | unsigned short |
| Description | A code representing the type of the underlying object. |

Example
var elem = document.getElementById('DDD');
var output1 = elem.nodeType;
var output2 = elem.firstChild.nodeType;

attribute: Document.parentNode (inherited from Node)

| Read-only | Yes |
| Type      | Node |
| Description | The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null. |

Example
var elem = document.getElementById('SSS');
var output = elem.parentNode.nodeName;

attribute: Document.childNodes (inherited from Node)

| Read-only | Yes |
| Type      | NodeList |

Example
var elem = document.getElementById('SSS');
var output = elem.childNodes;
**Description**  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

**NOTE**  The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

**Example**
```
var main = document.getElementById('doc');
var output1 = main.childNodes.length;
var output2 = main.childNodes[0].nodeType;
```

**attribute: **`Document.firstChild` **(inherited from Node)**

**Read-only**  Yes  
**Type**  Node  
**Description**  The first child of this node. If there is no such node, this returns null.

**Example**
```
var main = document.getElementById('doc');
var output = main.firstChild.nodeName;
```

**attribute: **`Document.lastChild` **(inherited from Node)**

**Read-only**  Yes  
**Type**  Node  
**Description**  The last child of this node. If there is no such node, this returns null.
Example
var main = document.getElementById('doc');
var output = main.lastChild.nodeName;

attribute: `Document.previousSibling` (inherited from Node)

Read-only Yes
Type Node
Description The node immediately preceding this node. If there is no such node, this returns null.

Example
var elem = document.getElementById('SSS');
var output = elem.previousSibling.nodeValue;

attribute: `Document.nextSibling` (inherited from Node)

Read-only Yes
Type Node
Description The node immediately following this node. If there is no such node, this returns null.

Example
var elem = document.getElementById('SSS');
var output = elem.nextSibling.nodeValue;

attribute: `Document.attributes` (inherited from Node)

Read-only Yes
Type NamedNodeMap
Description A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
attribute: `Document.ownerDocument`  
(inherited from Node)

Read-only  Yes
Type  Document
Description  The `Document` object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document`, this is null.

Example
```javascript
var main = document.getElementById('doc');
var output = main.ownerDocument.documentElement.nodeName;
```

method: `Document.createElement`  
(tagName)

Description  Creates an element of the type specified. Note that the instance returned implements the Element interface, so attributes can be specified directly on the returned object. In addition, if there are known attributes with default values, `Attr` nodes representing them are automatically created and attached to the element.

Parameters  `DOMString tagName`—The name of the element type to instantiate. (Remember that for XML this is case-sensitive.)

Returns  `Element`—A new `Element` object.

Exception

`DOMException INVALID_CHARACTER_ERR`  Raised if the specified name contains an invalid character.
Example

```javascript
var elem = document.createElement('P');
var output = elem.nodeName;
```

**method: `Document.createDocumentFragment()`**

<table>
<thead>
<tr>
<th>Description</th>
<th>Creates an empty DocumentFragment object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>None.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>None.</td>
</tr>
</tbody>
</table>

Example

```javascript
var main = document.getElementById('doc');
var elem = document.createDocumentFragment();
var nodeFor = main.firstChild;
elem.appendChild(nodeFor);
var output = elem.firstChild.nodeName;
```

**method: `Document.createTextNode(data)`**

<table>
<thead>
<tr>
<th>Description</th>
<th>Creates a Text node given the specified string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>DOMString—String data for the node.</td>
</tr>
<tr>
<td>Returns</td>
<td>Text—The new Text object.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>None.</td>
</tr>
</tbody>
</table>

Example

```javascript
var elem = document.createTextNode('This is some text. ');
var output = elem.nodeValue;
```

**method: `Document.createComment(data)`**

<table>
<thead>
<tr>
<th>Description</th>
<th>Creates a Comment node given the specified string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>DOMString—String data for the node.</td>
</tr>
</tbody>
</table>

Example

```javascript
var elem = document.createComment('This is some text. ');
```
Returns  Comment—The new Comment object.

Exceptions  None.

Example

```javascript
var comm = document.createComment('This is some comment.');
var output1 = comm.nodeType;
var output2 = comm.data;
```

**method: Document.createCDATASection(data)**

**Description**  Creates a CDATASection node whose value is the specified string.

**Parameters**  DOMString data—The data for the CDATASection contents.

**Returns**  CDATASection—The new CDATASection object.

**Exception**

DOMException NOT_SUPPORTED_ERR  Raised if this document is an HTML document.

Example

```javascript
var cSection = document.createCDATASection('some text for CDATA-section');
var output1 = cSection.nodeType;
var output2 = cSection.data;
```

**method: Document.createProcessingInstruction(target, data)**

**Description**  Creates a ProcessingInstruction node given the specified name and data strings.

**Parameters**  DOMString target—The target part of the processing instruction. DOMString data—The data for the node.

**Returns**  ProcessingInstruction—The new ProcessingInstruction object.
Exceptions

DOMException INVALID_CHARACTER_ERR  Raised if an invalid character is specified.

DOMException NOT_SUPPORTED_ERR  Raised if this document is an HTML document.

Example

```javascript
var pi = document.createProcessingInstruction('php', 'echo("another example")');
var output1 = pi.nodeType;
var output2 = pi.target;
var output3 = pi.data;
```

method: Document.createAttribute(name)

Description  Creates an Attr of the given name. Note that the Attr instance can then be set on an Element using the setAttribute() method.

Parameters  DOMString name—The name of the attribute.

Returns  Attr—A new Attr object.

Exception

DOMException INVALID_CHARACTER_ERR  Raised if the specified name contains an invalid character.

Example

```javascript
var main = document.getElementById('doc');
var attr = document.createAttribute('temp');
attr.value = 'temporary';
main.setAttributeNode(attr);
var output = main.getAttribute('temp');
```

method: Document.createEntityReference(name)

Description  Creates an EntityReference object. In addition, if the referenced entity is known, the child list of the
EntityReference node is made the same as that of the corresponding Entity node.

**Parameters**
- DOMString name—The name of the entity to reference.

**Returns**
- EntityReference—The new EntityReference object.

**Exceptions**
- DOMException INVALID_CHARACTER_ERR Raised if the specified name contains an invalid character.
- DOMException NOT_SUPPORTED_ERR Raised if this document is an HTML document.

**method: Document.getElementsByTagName(tagName)**

**Description**
Returns a NodeList containing all Elements of the given name in the same order as they appear in the source document.

**Parameters**
- DOMString tagName—The name of the tag to match on. The special value * matches all tags.

**Returns**
- NodeList—A new NodeList object containing all the matched Elements.

**Exceptions**
- None.

**Example**
```javascript
var collection = document.getElementsByTagName('DIV');
var output = collection[1].getAttribute('id');
```

**method: Document.insertBefore(newChild, refChild) (inherited from Node)**

**Description**
 Inserts the node newChild before the existing child node refChild. If refChild is null, inserts newChild at the end of the list of children.
**Parameters**
Node `newChild`—The node to insert.
Node `refChild`—The reference node, i.e., the node before which the new node must be inserted.

**Returns**
Node—The node being inserted.

**Exceptions**

**DOMException HIERARCHY_REQUEST_ERR**
Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors.

**DOMException WRONG_DOCUMENT_ERR**
Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**
Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**
Raised if `refChild` is not a child of this node.

**NOTE**
If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed.

**Example**
```javascript
var main = document.getElementById('doc');
main.insertBefore(document.getElementById('DDD'), main.firstChild);
var output = main.firstChild.firstChild.nodeValue;
```

**method: Document.replaceChild(newChild, oldChild) (inherited from Node)**

**Description**
Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed.
Parameters  Node newChild—The new node to put in the child list. Node oldChild—The node being replaced in the list.

Returns  Node—The node replaced.

Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

```javascript
var elem = document.getElementById('DDD');

elem.replaceChild(document.getElementById('SSS').firstChild, elem.firstChild);
var output = elem.firstChild.nodeValue;
```

method: Document.removeChild(oldChild) (inherited from Node)

Description  Removes the child node indicated by oldChild from the list of children and returns it.

Parameters  Node oldChild—The node being removed.

Returns  Node—The node removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.
DOM Appendix

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

```javascript
var elem = document.getElementById('SSS');
var output1 = elem.firstChild.nodeValue;
elem.removeChild(elem.firstChild);
var output2 = elem.firstChild;
```

method: `Document.appendChild(newChild)` (inherited from Node)

Description  Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

Parameters  Node `newChild`—The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

Returns  Node—The node added.

Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if `newChild` was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

```javascript
var main = document.getElementById('doc');
main.appendChild(main.childNodes[1]);
var output = main.lastChild.firstChild.nodeValue;
```
method: `Document.hasChildNodes()`  
(inherited from Node)

**Description**  This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  None.

**Returns**  Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  None.

**Example**
```javascript
var main = document.getElementById('doc');
var output1 = main.hasChildNodes();
var output2 = main.firstChild.hasChildNodes();
```

method: `Document.cloneNode(deep)`  
(inherited from Node)

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns null).

**Parameters**  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

**NOTE**
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.
**Example**

```javascript
var main = document.getElementById('doc');
var elem = main.childNodes[3];
var returns = elem.cloneNode(true);
var output = returns.firstChild.nodeValue;
```

**DOCUMENTFRAGMENT**

**Type of interface**  Fundamental

**Own properties**

- **Attributes**  None.
- **Methods**  None.

**Inherited properties**

- **Attributes**  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument
- **Methods**  insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

**Description**  For some cases (extracting a portion of a document’s tree, creating a new fragment of a document, inserting nodes as children of another node), it is desirable to have an object that can hold fragments of document. The Document object itself is a little bit heavyweight for this; something lightweight that behaves like Node is much better. When a DocumentFragment is inserted into a document (or indeed any other Node that may take children), the children of the DocumentFragment and not the DocumentFragment itself are inserted into the Node.

**Example: Source**

```html
<div id="doc">
  <div>
    Text in the first DIV.
  </div>
  <div id="DDD" class="secondClass">
    Some text in the second DIV.
  </div>
</div>
```
<div class='thirdClass'>
    Some text and <span id='SSS'>element</span> in the third DIV.
</div>
</div>

We can try <i>another element</i>. It will be much more <b>interesting</b>.
</div>
</div>

Text in the last DIV.
</div>
</div>

attribute: `DocumentFragment.nodeName`
(inherited from Node)

Read-only Yes
Type DOMString
Description The name of this node, depending on its type.

Example

<!-- See example under Document interface -->

attribute: `DocumentFragment.nodeValue`
(inherited from Node)

Read-only No
Type DOMString
Description The value of this node, depending on its type.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.
Example
<!-- See example under Document interface -->

attribute: DocumentFragment.nodeType
(inherited from Node)

Read-only  Yes
Type  unsigned short
Description  A code representing the type of the underlying object.

Example
<!-- See example under Document interface -->

attribute: DocumentFragment.parentNode
(inherited from Node)

Read-only  Yes
Type  Node
Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example
<!-- See example under Document interface -->

attribute: DocumentFragment.childNodes
(inherited from Node)

Read-only  Yes
Type  NodeList
Description  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.
NOTE
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

Example
<!-- See example under Document interface -->

attribute: DocumentFragment.firstChild
(inherited from Node)

Read-only  Yes
Type  Node
Description  The first child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: DocumentFragment.lastChild
(inherited from Node)

Read-only  Yes
Type  Node
Description  The last child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->
attribute: `DocumentFragment.previousSibling` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately preceding this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: `DocumentFragment.nextSibling` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately following this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: `DocumentFragment.attributes` (inherited from Node)

- **Read-only**: Yes
- **Type**: NamedNodeMap
- **Description**: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Example
<!-- See example under Document interface -->
attribute: DocumentFragment.owner-Document (inherited from Node)

Read-only: Yes
Type: Document
Description: The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example
<!-- See example under Document interface -->

method: DocumentFragment.insertBefore (newChild, refChild) (inherited from Node)

Description: Inserts the node newChild before the existing child node refChild. If refChild is null, inserts newChild at the end of the list of children.

Parameters: Node newChild—The node to insert. Node refChild—The reference node, i.e., the node before which the new node must be inserted.

Returns: Node—The node being inserted.

Exceptions

DOMException HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR: Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR: Raised if this node is read-only.

DOMException NOT_FOUND_ERR: Raised if refChild is not a child of this node.
**NOTE**

If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

---

**Example**

<!-- See example under Document interface -->

**method: `DocumentFragment.replaceChild(newChild, oldChild)` (inherited from Node)**

**Description**

Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed. Node `newChild`—The new node to put in the child list. Node `oldChild`—The node being replaced in the list.

**Returns**

Node—The node replaced.

**Exceptions**

- **DOMException HIERARCHY_REQUEST_ERR** Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node's ancestors.

- **DOMException WRONG_DOCUMENT_ERR** Raised if `newChild` was created from a different document than the one that created this node.

- **DOMException NO_MODIFICATION_ALLOWED_ERR** Raised if this node is read-only.

- **DOMException NOT_FOUND_ERR** Raised if `oldChild` is not a child of this node.

**Example**

<!-- See example under Document interface -->
method: `DocumentFragment.removeChild(oldChild)` (inherited from `Node`)

**Description**
Removes the child node indicated by `oldChild` from the list of children and returns it.

**Parameters**
- `Node oldChild`—The node being removed.

**Returns**
- `Node`—The node removed.

**Exceptions**
- `DOMException NO_MODIFICATION_ALLOWED_ERR` Raised if this node is read-only.
- `DOMException NOT_FOUND_ERR` Raised if `oldChild` is not a child of this node.

**Example**
```
<!-- See example under Document interface -->
```

method: `DocumentFragment.appendChild(newChild)` (inherited from `Node`)

**Description**
Adds the node `newChild` to the end of the list of children of this node. If `newChild` is already in the tree, it is first removed.

**Parameters**
- `Node newChild`—The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

**Returns**
- `Node`—The node added.

**Exceptions**
- `DOMException HIERARCHY_REQUEST_ERR` Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.
DOMException WRONGDOCUMENT_ERR
Raised if newChild was created from a different document than the one that created this node.

DOMException NOMODIFICATIONALLOWED_ERR
Raised if this node is read-only.

Example
<!-- See example under Document interface -->

method: DocumentFragment.hasMoreChildren() (inherited from Node)

Description  This is a convenient method to allow easy determination of whether a node has any children.

Parameters  None.

Returns  Boolean—True if the node has any children, false if the node has no children.

Exceptions  None.

Example
<!-- See example under Document interface -->

method: DocumentFragment.cloneNode(deep) (inherited from Node)

Description  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

Parameters  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

Returns  Node—The duplicate node.

Exceptions  None.
NOTE

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example

<!-- See example under Document interface -->

DOCUMENTTYPE

Type of interface   Extended

Own properties

Attributes   name, entities, notations

Methods   None.

Inherited properties

Attributes   nodeName, nodeValue, nodeType, parentNode,.childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods   insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

Description   The document's <!DOCTYPE> element is reflected in the Document.doctype attribute. The value of this attribute is either null or the DocumentType object.

attribute: DocumentType.name

Read-only   Yes

Type   DOMString

Description   The name of the DTD; i.e., the name immediately following the DOCTYPE keyword.
attribute: `DocumentType.entities`

Read-only: Yes
Type: NamedNodeMap

Description: A NamedNodeMap containing the general entities, both external and internal, declared in the DTD. Duplicates are discarded.

attribute: `DocumentType.notations`

Read-only: Yes
Type: NamedNodeMap

Description: A NamedNodeMap containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the Notation interface.

attribute: `DocumentType.nodeName` (inherited from Node)

Read-only: Yes
Type: DOMString

Description: The name of this node, depending on its type.

Example

<!-- See example under Document interface -->

attribute: `DocumentType.nodeValue` (inherited from Node)

Read-only: No
Type: DOMString

Description: The value of this node, depending on its type.
Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example
<!-- See example under Document interface -->

attribute: DocumentType.nodeType
(inherited from Node)

Read-only  Yes
Type  unsigned short
Description  A code representing the type of the underlying object.

Example
<!-- See example under Document interface -->

attribute: DocumentType.parentNode
(inherited from Node)

Read-only  Yes
Type  Node
Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example
<!-- See example under Document interface -->
attribute: `DocumentType.childNodes`  
(inherited from `Node`)  

**Read-only** Yes  
**Type** NodeList  
**Description** A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

**NOTE**  
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the `getElementsByTagName()` method.

**Example**  
`<!-- See example under Document interface -->`

attribute: `DocumentType.firstChild`  
(inherited from `Node`)  

**Read-only** Yes  
**Type** Node  
**Description** The first child of this node. If there is no such node, this returns null.

attribute: `DocumentType.lastChild`  
(inherited from `Node`)  

**Read-only** Yes  
**Type** Node  
**Description** The last child of this node. If there is no such node, this returns null.
attribute: `DocumentType.previousSibling`  
* (inherited from Node)*

Read-only: Yes  
Type: Node  
Description: The node immediately preceding this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->

attribute: `DocumentType.nextSibling`  
* (inherited from Node)*

Read-only: Yes  
Type: Node  
Description: The node immediately following this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->

attribute: `DocumentType.attributes`  
* (inherited from Node)*

Read-only: Yes  
Type: NamedNodeMap  
Description: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Example

<!-- See example under Document interface -->
attribute: `DocumentType.ownerDocument`
(inherited from Node)

**Read-only** Yes

**Type** Document

**Description** The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

**Example**

<!-- See example under Document interface -->

method: `DocumentType.insertBefore`
(newChild, refChild)
(inherited from Node)

**Description** Inserts the node newChild before the existing child node refChild. If refChild is null, inserts newChild at the end of the list of children.

**Parameters**
- `Node newChild`—The node to insert.
- `Node refChild`—The reference node, i.e., the node before which the new node must be inserted.

**Returns** `Node`—The node being inserted.

**Exceptions**

- **DOMException HIERARCHY_REQUEST_ERR** Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node’s ancestors.

- **DOMException WRONG_DOCUMENT_ERR** Raised if newChild was created from a different document than the one that created this node.

- **DOMException NO_MODIFICATION_ALLOWED_ERR** Raised if this node is read-only.

- **DOMException NOT_FOUND_ERR** Raised if refChild is not a child of this node.
NOTE

If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

Example

<!-- See example under Document interface -->

**method: DocumentType.replaceChild**

*(newChild, oldChild) (inherited from Node)*

**Description**

Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed.

**Parameters**

Node `newChild`—The new node to put in the child list. Node `oldChild`—The node being replaced in the list.

**Returns**

Node—The node replaced.

**Exceptions**

DOMException `HIERARCHY_REQUEST_ERR` Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node’s ancestors.

DOMException `WRONG_DOCUMENT_ERR` Raised if `newChild` was created from a different document than the one that created this node.

DOMException `NO_MODIFICATION_ALLOWED_ERR` Raised if this node is read-only.

DOMException `NOT_FOUND_ERR` Raised if `oldChild` is not a child of this node.

Example

<!-- See example under Document interface -->
method: `DocumentType.removeChild(oldChild)` (inherited from Node)

Description  Removes the child node indicated by `oldChild` from the list of children and returns it.

Parameters  Node `oldChild`—The node being removed.

Returns  Node—The node removed.

Exceptions  
- `DOMException NO_MODIFICATION_ALLOWED_ERR` Raised if this node is read-only.
- `DOMException NOT_FOUND_ERR` Raised if `oldChild` is not a child of this node.

Example

```html
<!-- See example under Document interface -->
```

method: `DocumentType.appendChild(newChild)` (inherited from Node)

Description  Adds the node `newChild` to the end of the list of children of this node. If `newChild` is already in the tree, it is first removed.

Parameters  Node `newChild`—The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

Returns  Node—The node added.

Exceptions  
- `DOMException HIERARCHY_REQUEST_ERR` Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.
DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

**Example**

<!-- See example under Document interface -->

**method: DocumentType. hasChildNodes()**

*(inherited from Node)*

**Description**  This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  None.

**Returns**  Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  None.

**Example**

<!-- See example under Document interface -->

**method: DocumentType. cloneNode(deep)**

*(inherited from Node)*

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

**Parameters**  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.
NOTE
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example
<!-- See example under Document interface -->

DOMImplementation

Type of interface Fundamental

Own properties
Attributes None.
Methods hasFeature

Description The DOMImplementation interface provides a number of methods for performing operations that are independent of any particular instance of the document object model. (The DOM Level 1 does not specify a way of creating a document instance, and hence document creation is an operation specific to an implementation. Future Levels of the DOM specification are expected to provide methods for creating documents directly.)

method: DOMImplementation.hasFeature (feature, version)

Description Tests if the DOM implementation implements a specific feature.

Parameters DOMString feature—The package name of the feature to test. In Level 1, the legal values are “HTML” and “XML” (case-insensitive). DOMString version—This is the version number of the package name to test. In Level 1, version 1.0, this is the string “1.0”. If the version is not specified, supporting any version of the feature will cause the method to return true.
**Returns**  Boolean—True if the feature is implemented in the specified version, false otherwise.

**Exceptions**  None.

**Example**

```javascript
var output = document.implementation.hasFeature('HTML', '1.0');
```

## Element

**Type of interface**  Fundamental

**Own properties**

- **Attributes**  `tagName`
- **Methods**  `getAttribute`, `setAttribute`, `removeAttribute`, `getAttributeNode`, `setAttributeNode`, `removeAttributeNode`, `getElementsByTagName`, `normalize`

**Inherited properties**

- **Attributes**  `nodeName`, `nodeValue`, `nodeType`, `parentNode`, `childNodes`, `firstChild`, `lastChild`, `previousSibling`, `nextSibling`, `attributes`, `ownerDocument`
- **Methods**  `insertBefore`, `replaceChild`, `removeChild`, `appendChild`, `hasChildNodes`, `cloneNode`

**Description**  Apart from text, Element nodes are the most common objects in every XML document.

### attribute: `Element.tagName`

- **Read-only**  Yes
- **Type**  DOMString
- **Description**  The name of the element.

**Example**

```javascript
var main = document.getElementById('doc');
var output = main.tagName;
```
attribute: `Element.nodeName` (inherited from Node)

- **Read-only**: Yes
- **Type**: DOMString
- **Description**: The name of this node, depending on its type.

Example

<!-- See example under Document interface -->

attribute: `Element.nodeValue` (inherited from Node)

- **Read-only**: No
- **Type**: DOMString
- **Description**: The value of this node, depending on its type.

Exceptions

- **DOMException NO_MODIFICATION_ALLOWED_ERR**
  This exception raises on setting when the node is read-only.

- **DOMException DOMSTRING_SIZE_ERR**
  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

<!-- See example under Document interface -->

attribute: `Element.nodeType` (inherited from Node)

- **Read-only**: Yes
- **Type**: unsigned short
- **Description**: A code representing the type of the underlying object.
attribute: `Element.parentNode` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node

**Description**: The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

NOTE
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

attribute: `Element.childNodes` (inherited from Node)

- **Read-only**: Yes
- **Type**: NodeList

**Description**: A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.
attribute: `Element.firstChild` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The first child of this node. If there is no such node, this returns null.

**Example**

`<!-- See example under Document interface -->`

attribute: `Element.lastChild` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The last child of this node. If there is no such node, this returns null.

**Example**

`<!-- See example under Document interface -->`

attribute: `Element.previousSibling` (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately preceding this node. If there is no such node, this returns null.

**Example**

`<!-- See example under Document interface -->`
attribute: `Element.nextIntSibling` (inherited from Node)

Read-only Yes
Type Node
Description The node immediately following this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: `Element.attributes` (inherited from Node)

Read-only Yes
Type NamedNodeMap
Description A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Example
<!-- See example under Document interface -->

attribute: `Element.ownerDocument` (inherited from Node)

Read-only Yes
Type Document
Description The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example
<!-- See example under Document interface -->
method: `Element.getAttribute(name)`

**Description**
Retrieves an attribute value by name.

**Parameters**
- `DOMString name`—The name of the attribute to retrieve.

**Returns**
- `DOMString`—The Attr value as a string, or the empty string if that attribute does not have a specified or default value.

**Exceptions**
None.

**Example**
```javascript
var main = document.getElementById('doc');
var output1 = main.getAttribute('id');
var output2 = main.getAttribute('class');
```

method: `Element.setAttribute(name, value)`

**Description**
Adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string; it is not parsed as it is being set. Therefore, any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an Attr node plus any Text and EntityReference nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.

**Parameters**
- `DOMString name`—The name of the attribute to create or alter.
- `DOMString value`—Value to set in string form.

**Returns**
Nothing.

**Exceptions**
- `DOMException INVALID_CHARACTER_ERR` Raised if the specified name contains an invalid character.
DOMException NO_MODIFICATION_ALLOWED_ERR
Raised if this node is read-only.

Example
```
var main = document.getElementById('doc');
main.setAttribute('attr', 'temp');
var output = main.getAttribute('attr');
```

method: `Element.removeAttribute(name)`

Description
Removes an attribute by name. If the removed attribute is known to have a default value, an attribute immediately appears containing the default value.

Parameters
DOMString name—The name of the attribute to remove.

Returns
Nothing.

Example
```
var main = document.getElementById('doc');
main.removeAttribute('id');
var output = main.getAttribute('id');
```

method: `Element.getAttributeNode(name)`

Description
Retrieves an Attr node by name.

Parameters
DOMString name—The name of the attribute to retrieve.

Returns
Attr—The Attr node with the specified attribute name or null if there is no such attribute.

Exceptions
None.
Example

```javascript
var main = document.getElementById('doc');
var output = main.getAttributeNode('id').value;
```

**method: Element.setAttributeNode (newAttr)**

**Description**  Adds a new attribute. If an attribute with that name is already present in the element, it is replaced by the new one.

**Parameters**  Attr newAttr—The Attr node to add to the attribute list.

**Returns**  Attr—If the newAttr attribute replaces an existing attribute with the same name, the previously existing Attr node is returned; otherwise null is returned.

**Exceptions**

DOMException WRONG_DOCUMENT_ERR  Raised if newAttr was created from a different document than the one that created the element.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException INUSE_ATTRIBUTE_ERR  Raised if newAttr is already an attribute of another Element object. The DOM user must explicitly clone Attr nodes to reuse them in other elements.

Example

```javascript
var collection = document.getElementsByTagName('div');
var elem = collection[0];
var attr = collection[2].getAttributeNode('class');
elem.setAttributeNode(attr);
var output = elem.getAttributeNode('class').value;
```
method: `Element.removeAttributeNode(oldAttr)`

**Description**  
Removes the specified attribute.

**Parameters**  
- `oldAttr`—The `Attr` node to remove from the attribute list. If the removed `Attr` has a default value, it is immediately replaced.

**Returns**  
- `Attr`—The `Attr` node that was removed.

**Exceptions**

- **DOMException NO_MODIFICATION_ALLOWED_ERR**  
  Raised if this node is read-only.

- **DOMException NOT_FOUND_ERR**  
  Raised if `oldAttr` is not an attribute of the element.

**Example**

```javascript
var main = document.getElementById('doc');
var attrOut = main.removeAttributeNode(attr);
var output1 = main.getAttributeNode('id');
var output2 = attrOut.value;
```

method: `Element.getElementsByTagName(name)`

**Description**  
Returns a NodeList containing all Elements of the given name in the same order as they appear in the source document.

**Parameters**  
- `DOMString name`—The name of the tag to match on. The special value `*` matches all tags.

**Returns**  
- `NodeList`—A list of matching Element nodes.

**Exceptions**  
None.

**Example**

```javascript
var main = document.getElementById('doc');
var collection = main.getElementsByTagName('div');
var output1 = collection.length;
var output2 = collection[0].firstChild.nodeValue;
```
method: `Element.normalize()`

**Description**  Puts all Text nodes in the full depth of the sub-tree underneath this Element into a “normal” form where only markup (e.g., tags, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are no adjacent Text nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and reloaded, and is useful when operations (such as XPointer lookups) that depend on a particular document tree structure are to be used.

**Parameters**  None.

**Returns**  Nothing.

**Exceptions**  None.

---

Example: Source

```
<div id="doc"></div>
```

---

Example

```javascript
var main = document.getElementById('doc');
var output1 = main.childNodes.length;
var textNode1 = document.createTextNode('This is some
text.');</
var textNode2 = document.createTextNode('This is another
text.');</
main.appendChild(textNode1);
main.appendChild(textNode2);
var output2 = main.childNodes.length;
main.normalize();
var output3 = main.childNodes.length;
```

method: `Element.insertBefore(newChild, refChild)` (inherited from Node)

**Description**  Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.
Parameters

Node newChild—The node to insert. Node refChild—The reference node, i.e., the node before which the new node must be inserted.

Returns

Node—The node being inserted.

Exceptions

DOMException HIERARCHY_REQUEST_ERR Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node's ancestors.

DOMException WRONG_DOCUMENT_ERR Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

DOMException NOT_FOUND_ERR Raised if refChild is not a child of this node.

NOTE

If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If newChild is already in the tree, it is first removed.

Example

<!-- See example under Document interface -->

method: Element.replaceChild(newChild, oldChild) (inherited from Node)

Description

Replaces the child node oldChild with newChild in the list of children and returns the oldChild node. If newChild is already in the tree, it is first removed.

Parameters

Node newChild—The new node to put in the child list. Node oldChild—The node being replaced in the list.

Returns

Node—The node replaced.
Exceptions

**DOMException HIERARCHY_REQUEST_ERR**  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR**  Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  Raised if `oldChild` is not a child of this node.

Example

<!-- See example under Document interface -->

**method: Element.removeChild(oldChild)**  (inherited from Node)

**Description**  Removes the child node indicated by `oldChild` from the list of children and returns it.

**Parameters**  Node `oldChild`—The node being removed.

**Returns**  Node—The node removed.

Exceptions

**DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  Raised if `oldChild` is not a child of this node.

**method: Element.appendChild(newChild)**  (inherited from Node)

**Description**  Adds the node `newChild` to the end of the list of children of this node. If `newChild` is already in the tree, it is first removed.
Parameters  Node newChild—The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

Returns  Node—The node added.

Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

<!-- See example under Document interface -->

**method:** Element.hasChildNodes()

(inherited from Node)

**Description**  This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  None.

**Returns**  Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  None.

Example

<!-- See example under Document interface -->
**method: Element.cloneNode(deep)**
(inherited from Node)

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

**Parameters**  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

**NOTE**
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

**Example**
<!-- See example under Document interface -->

**ENTITY**

**Type of interface**  Extended

**Own properties**

**Attributes**  publicId, systemId, notationName

**Methods**  None.

**Inherited properties**

**Attributes**  nodeName,nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

**Methods**  insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode
Description  This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself not the entity declaration. Entity declaration modeling has been left for a later level of the DOM specification.

attribute: **Entity.publicId**

Read-only  Yes
Type  DOMString
Description  Stores the public identifier associated with the Entity (or null if the identifier was not specified).

attribute: **Entity.systemId**

Read-only  Yes
Type  DOMString
Description  Stores the system identifier associated with the Entity (or null if the identifier was not specified).

attribute: **Entity.notationName**

Read-only  Yes
Type  DOMString
Description  For unparsed entities, the name of the Notation for the Entity. For parsed entities, this is null.

attribute: **Entity.nodeName** (inherited from Node)

Read-only  Yes
Type  DOMString
Description  The name of this node, depending on its type.

Example

<!-- See example under Document interface -->
**attribute: Entity.nodeValue (inherited from Node)**

Read-only  No
Type  DOMString
Description  The value of this node, depending on its type.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

**Example**

<!-- See example under Document interface -->

**attribute: Entity.nodeType (inherited from Node)**

Read-only  Yes
Type  unsigned short
Description  A code representing the type of the underlying object.

**Example**

<!-- See example under Document interface -->

**attribute: Entity.parentNode (inherited from Node)**

Read-only  Yes
Type  Node

**Example**

<!-- See example under Document interface -->
Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example

<!-- See example under Document interface -->

attribute: Entity.childNodes (inherited from Node)

Read-only  Yes
Type    NodeList
Description  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

NOTE
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

Example

<!-- See example under Document interface -->

attribute: Entity.firstChild (inherited from Node)

Read-only  Yes
Type    Node
Description  The first child of this node. If there is no such node, this returns null.
attribute: **Entity.lastChild** (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The last child of this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->

attribute: **Entity.previousSibling** (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately preceding this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->

attribute: **Entity.nextSibling** (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately following this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->
attribute: `Entity.attributes` (inherited from Node)

Read-only: Yes

Type: NamedNodeMap

Description: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Example

<!-- See example under Document interface -->

attribute: `Entity.ownerDocument` (inherited from Node)

Read-only: Yes

Type: Document

Description: The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example

<!-- See example under Document interface -->

method: `Entity.insertBefore(newChild, refChild)` (inherited from Node)

Description: Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.

Parameters

- `newChild`: The node to insert. Node
- `refChild`: The reference node, i.e., the node before which the new node must be inserted.

Returns: Node—The node being inserted.
Exceptions

DOMException 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 insert is one of this node’s ancestors.

DOMException WRONG_DOCUメント_ERR  Raised if newNode was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if refChild is not a child of this node.

NOTE
If newNode is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If newNode is already in the tree, it is first removed.

Example
<!-- See example under Document interface -->

method: Entity.replaceChild(newChild, oldChild) (inherited from Node)

Description  Replaces the child node oldChild with newNode in the list of children and returns the oldChild node. If newNode is already in the tree, it is first removed.

Parameters  Node newNode—The new node to put in the child list. Node oldChild—The node being replaced in the list.

Returns  Node—The node replaced.

Exceptions

DOMException 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.
DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

<!-- See example under Document interface -->

method: Entity.removeChild(oldChild)
(inherited from Node)

Description  Removes the child node indicated by oldChild from the list of children and returns it.

Parameters  Node oldChild—The node being removed.

Returns  Node—The node removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

<!-- See example under Document interface -->

method: Entity.appendChild(newChild)
(inherited from Node)

Description  Adds the node newChild to the end of the list of children of this node. If newChild is already in the tree, it is first removed.
**Parameters**  
Node newChild—The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

**Returns**  
Node—The node added.

**Exceptions**  

DOMException HIERARCHY_REQUEST_ERR  
Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors.

DOMException WRONG_DOCUMENT_ERR  
Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  
Raised if this node is read-only.

**Example**  
<!-- See example under Document interface -->

**method: Entity.hasChildNodes() (inherited from Node)**  

**Description**  
This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  
None.

**Returns**  
Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  
None.

**Example**  
<!-- See example under Document interface -->
method: Entity.cloneNode(deep) (inherited from Node)

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

**Parameters**  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

**NOTE**
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

**Example**

<!-- See example under Document interface -->

---

**ENTITY REFERENCE**

**Type of interface**  Extended

**Own properties**

**Attributes**  None.

**Methods**  None.

**Inherited properties**

**Attributes**  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

**Methods**  insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode
Description  EntityReferences contains the name of the Entity from <!ENTITY foo SYSTEM "foo.xml">. Please note that character references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference.

attribute: EntityReference.nodeName
(inherited from Node)

Read-only  Yes
Type  DOMString
Description  The name of this node, depending on its type.

Example
<!-- See example under Document interface -->

attribute: EntityReference.nodeValue
(inherited from Node)

Read-only  No
Type  DOMString
Description  The value of this node, depending on its type.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example
<!-- See example under Document interface -->
attribute: `EntityReference.nodeType`  
(inherited from Node)

- **Read-only**: Yes
- **Type**: unsigned short
- **Description**: A code representing the type of the underlying object.

**Example**

<!-- See example under Document interface -->

attribute: `EntityReference.parentNode`  
(inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

**Example**

<!-- See example under Document interface -->

attribute: `EntityReference.childNodes`  
(inherited from Node)

- **Read-only**: Yes
- **Type**: NodeList
- **Description**: A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.
NOTE
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

Example
<!-- See example under Document interface -->

attribute: EntityReference.firstChild
(inherited from Node)

Read-only  Yes
Type  Node
Description  The first child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: EntityReference.lastChild
(inherited from Node)

Read-only  Yes
Type  Node
Description  The last child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->
**attribute: EntityReference.previousSibling**  
(inherited from Node)

Read-only: Yes  
Type: Node  
Description: The node immediately preceding this node. If there is no such node, this returns null.

**Example**

<!-- See example under Document interface -->

**attribute: EntityReference.nextSibling**  
(inherited from Node)

Read-only: Yes  
Type: Node  
Description: The node immediately following this node. If there is no such node, this returns null.

**Example**

<!-- See example under Document interface -->

**attribute: EntityReference.attributes**  
(inherited from Node)

Read-only: Yes  
Type: NamedNodeMap  
Description: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

**Example**

<!-- See example under Document interface -->
attribute: `EntityReference.ownerDocument` (inherited from `Node`)

- **Read-only**: Yes
- **Type**: Document
- **Description**: The `Document` object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document`, this is null.

Example

<!-- See example under Document interface -->

method: `EntityReference.insertBefore` *(newChild, refChild)* (inherited from `Node`)

- **Description**: Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.

- **Parameters**: Node `newChild`—The node to insert. Node `refChild`—The reference node, i.e., the node before which the new node must be inserted.

- **Returns**: Node—The node being inserted.

Exceptions

- **DOMException HIERARCHY_REQUEST_ERR**: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node’s ancestors.

- **DOMException WRONG_DOCUMENT_ERR**: Raised if `newChild` was created from a different document than the one that created this node.

- **DOMException NO_MODIFICATION_ALLOWED_ERR**: Raised if this node is read-only.

- **DOMException NOT_FOUND_ERR**: Raised if `refChild` is not a child of this node.
NOTE
If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

Example

<!-- See example under Document interface -->

method: `EntityReference.replaceChild(newChild, oldChild)` (inherited from `Node`)

**Description**  Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If `newChild` is already in the tree, it is first removed.

**Parameters**  Node `newChild`—The new node to put in the child list. Node `oldChild`—The node being replaced in the list.

**Returns**  Node—The node replaced.

**Exceptions**

**DOMException HIERARCHY_REQUEST_ERR**  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR**  Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  Raised if `oldChild` is not a child of this node.

Example

<!-- See example under Document interface -->
method: `EntityReference.removeChild(oldChild)` (inherited from `Node`)

**Description**
Removes the child node indicated by `oldChild` from the list of children and returns it.

**Parameters**
- `Node oldChild`—The node being removed.

**Returns**
- `Node`—The node removed.

**Exceptions**
- **DOMException NO_MODIFICATION_ALLOWED_ERR**
  Raised if this node is read-only.

- **DOMException NOT_FOUND_ERR**
  Raised if `oldChild` is not a child of this node.

**Example**

```
<!-- See example under Document interface -->
```
DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example
<!-- See example under Document interface -->

method: EntityReference.hasChildNodes() (inherited from Node)

Description  This is a convenient method to allow easy determination of whether a node has any children.

Parameters  None.

Returns  Boolean—True if the node has any children, false if the node has no children.

Exceptions  None.

Example
<!-- See example under Document interface -->

method: EntityReference.cloneNode(deep) (inherited from Node)

Description  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

Parameters  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

Returns  Node—The duplicate node.

Exceptions  None.
NOTE

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example

<!-- See example under Document interface -->

**NamedNodeMap**

**Type of interface**  Fundamental

**Own properties**

- **Attributes**  length
- **Methods**  getNamedItem, setNamedItem, removeNamedItem, item

**Description**  Objects implementing the NamedNodeMap interface are used to represent collections of nodes that can be accessed by name. NamedNodeMap are not maintained in any particular order.

**attribute: NamedNodeMap.length**

- **Read-only**  Yes
- **Type**  unsigned long

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

Example

```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var output = attrNode.length;
```
method: `NamedNodeMap.getNamedItem(name)`

**Description**
Retrieves a node specified by name.

**Parameters**
DOMString `name`—Name of a node to retrieve.

**Returns**
Node—A Node (of any type) with the specified name, or null if the specified name did not identify any node in the map.

**Exceptions**
None.

**Example**
```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var output = attrNode.getNamedItem('class').nodeValue;
```

method: `NamedNodeMap.setNamedItem(arg)`

**Description**
Adds a node using its `nodeName` attribute.

**Parameters**
Node arg—A node to store in a named node map. If a node with that name is already present in the map, it is replaced by the new one.

**Returns**
Node—If the new Node replaces an existing node with the same name, the previously existing Node is returned; otherwise null is returned.

**Exceptions**
`DOMException WRONG_DOCUMENT_ERR` Raised if arg was created from a different document than the one that created the `NamedNodeMap`.

`DOMException NO_MODIFICATION_ALLOWED_ERR` Raised if this `NamedNodeMap` is read-only.

`DOMException INUSE_ATTRIBUTE_ERR` Raised if arg is an `Attr` that is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` nodes to reuse them in other elements.
NOTE
Remember that multiple nodes of certain types (those that have a “special” string value) are not allowed.

Example

```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var attr = document.createAttribute('temp');
attr.value = 'temporary';
attrNode.setNamedItem(attr);
var output = attrNode.getNamedItem('temp').nodeValue;
```

**method: NamedNodeMap.removeNamedItem(name)**

**Description**
Removes a node specified by name. If the removed node is an Attr with a default value, it is immediately replaced.

**Parameters**
DOMString name—The name of a node to remove.

**Returns**
Node—The node removed from the map if a node with such a name exists.

**Exceptions**

DOMException NOT_FOUND_ERR
Raised if there is no node named name in the map.

DOMException NO_MODIFICATION_ALLOWED_ERR
Raised when the NamedNodeMap is read-only.

Example

```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var removedNode = attrNode.removeNamedItem('class');
var output1 = removedNode.nodeValue;
var output2 = attrNode.getNamedItem('class');
```
method: **NamedNodeMap.item(index)**

Description  Returns the indexth item in the map. If index is greater than or equal to the number of nodes in the map, this returns null.

Parameters  unsigned long index—Index into the map.

Returns  Node—The node at the indexth position in the NamedNodeMap, or null if that is not a valid index.

Exceptions  None.

Example

```javascript
var main = document.getElementById('doc');
var attrNode = main.childNodes[3].attributes;
var output = attrNode.item(0).nodeValue;
```

**Node**

Type of interface  Fundamental

Own properties

Attributes  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods  insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

Description  The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree.

**NOTE**

Node is every single part of the document—text, element, etc. See NodeTypes.

attribute: **Node.nodeType**

Read-only  Yes
122 DOM Appendix

Type DOMString
Description The name of this node, depending on its type.

Example
<!-- See example under Document interface -->

attribute: Node.nodeName

Read-only No
Type DOMString
Description The value of this node, depending on its type.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR
This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example
<!-- See example under Document interface -->

attribute: Node.nodeType

Read-only Yes
Type unsigned short
Description A code representing the type of the underlying object.

Example
<!-- See example under Document interface -->

attribute: Node.parentNode

Read-only Yes
Type  Node

Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example
<!-- See example under Document interface -->

attribute: Node.childNodes

Read-only  Yes

Type  NodeList

Description  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

NOTE  The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

Example
<!-- See example under Document interface -->

attribute: Node.firstChild

Read-only  Yes

Type  Node

Description  The first child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->
attribute: **Node.lastChild**

**Read-only**  Yes

**Type**  Node

**Description**  The last child of this node. If there is no such node, this returns null.

**Example**

<!-- See example under Document interface -->

attribute: **Node.previousSibling**

**Read-only**  Yes

**Type**  Node

**Description**  The node immediately preceding this node. If there is no such node, this returns null.

**Example**

<!-- See example under Document interface -->

attribute: **Node.nextSibling**

**Read-only**  Yes

**Type**  Node

**Description**  The node immediately following this node. If there is no such node, this returns null.

**Example**

<!-- See example under Document interface -->

attribute: **Node.attributes**

**Read-only**  Yes

**Type**  NamedNodeMap

**Description**  A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
attribute: Node.ownerDocument

Read-only: Yes

Type: Document

Description: The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example

<!-- See example under Document interface -->

method: Node.insertBefore(newChild, refChild)

Description: Inserts the node newChild before the existing child node refChild. If refChild is null, inserts newChild at the end of the list of children.

Parameters:
- Node newChild—The node to insert. Node
- refChild—The reference node, i.e., the node before which the new node must be inserted.

Returns: Node—The node being inserted.

Exceptions

DOMException HIERARCHY_REQUEST_ERR Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.
DOMException NOT_FOUND_ERR Raised if refChild is not a child of this node.

NOTE
If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If newChild is already in the tree, it is first removed.

Example
<!-- See example under Document interface -->

**method: Node.replaceChild(newChild, oldChild)**

**Description** Replaces the child node oldChild with newChild in the list of children and returns the oldChild node. If the newChild is already in the tree, it is first removed.

**Parameters** Node newChild—The new node to put in the child list. Node oldChild—The node being replaced in the list.

**Returns** Node—The node replaced.

**Exceptions**

DOMException HIERARCHY_REQUEST_ERR Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR Raised if this node is read-only.

DOMException NOT_FOUND_ERR Raised if oldChild is not a child of this node.

Example
<!-- See example under Document interface -->
method: `Node.removeChild(oldChild)`

**Description**  
Removes the child node indicated by `oldChild` from the list of children and returns it.

**Parameters**  
Node `oldChild` — The node being removed.

**Returns**  
Node — The node removed.

**Exceptions**

DOMException NO_MODIFICATION_ALLOWED_ERR  
Raised if this node is read-only.

DOMException NOT_FOUND_ERR  
Raised if `oldChild` is not a child of this node.

**Example**

<!-- See example under Document interface -->

method: `Node.appendChild(newChild)`

**Description**  
Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

**Parameters**  
Node `newChild` — The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

**Returns**  
Node — The node added.

**Exceptions**

DOMException HIERARCHY_REQUEST_ERR  
Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.

DOMException WRONGDOCUMENT_ERR  
Raised if `newChild` was created from a different document than the one that created this node.
DOM Exception NO_MODIFICATION_ALLOWED_ERR
Raised if this node is read-only.

Example
<!-- See example under Document interface -->

**method: Node.hasChildNodes()**

**Description**
This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**
None.

**Returns**
Boolean—True if the node has any children, false if the node has no children.

**Exceptions**
None.

Example
<!-- See example under Document interface -->

**method: Node.cloneNode(deep)**

**Description**
Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

**Parameters**
Boolean deep—If true, recursively clone the subtree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**
Node—The duplicate node.

**Exceptions**
None.

**NOTE**
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.
Example
<!-- See example under Document interface -->

**NODELIST**

*Type of interface*  Fundamental

**Own properties**

Attributes  length

Methods  item

**Description**  The NodeList interface is an ordered collection of nodes. Its items are accessible via an integral index, starting from 0.

**attribute: NodeList.length**

*Read-only*  Yes

*Type*  unsigned long

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

**Example**

```javascript
var main = document.getElementById('doc');
var collection = main.getElementsByTagName('DIV');
var output = collection.length;
```

**method: NodeList.item(index)**

*Description*  Returns the indexth item in the collection. If index is greater than or equal to the number of nodes in the list, this returns null.

*Parameters*  unsigned long index—Index into the collection.

*Returns*  Node—The node at the indexth position in the NodeList, or null if that is not a valid index.

*Exceptions*  None.
Example

```javascript
var collection = document.getElementsByTagName('DIV');
var output1 = collection.item(2).firstChild.nodeValue;
var output2 = collection[2].firstChild.nodeValue;
```

**NOTATION**

**Type of interface**  Extended

**Own properties**

- **Attributes**  publicId, systemId
- **Methods**  None.

**Inherited properties**

- **Attributes**  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument
- **Methods**  insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

**Description**  Notation is used to provide information needed to allow the application to call a processor for data in the notation described (<!NOTATION .. >).

**NOTE**

It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.

**attribute: Notation.publicId**

- **Read-only**  Yes
- **Type**  DOMString
- **Description**  Stores the public identifier of the Notation (or null if the identifier was not specified).
attribute: *Notation.systemId*

Read-only  Yes  
Type  DOMString  
Description  Stores the system identifier of the Notation (or null if the identifier was not specified).

attribute: *Notation.nodeName* (inherited from Node)

Read-only  Yes  
Type  DOMString  
Description  The name of this node, depending on its type.

attribute: *Notation.nodeValue* (inherited from Node)

Read-only  No  
Type  DOMString  
Description  The value of this node, depending on its type.

Exceptions

**DOMException NO_MODIFICATION_ALLOWED_ERR**  
This exception raises on setting when the node is read-only.

**DOMException DOMSTRING_SIZE_ERR**  
This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

<!-- See example under Document interface -->
attribute: Notation.nodeType (inherited from Node)

Read-only  Yes

Type  unsigned short

Description  A code representing the type of the underlying object.

Example
<!-- See example under Document interface -->

attribute: Notation.parentNode (inherited from Node)

Read-only  Yes

Type  Node

Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example
<!-- See example under Document interface -->

attribute: Notation.childNodes (inherited from Node)

Read-only  Yes

Type  NodeList

Description  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.
NOTE

The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

Example

<!-- See example under Document interface -->

attribute: Notation.firstChild (inherited from Node)

Read-only  Yes
Type  Node
Description  The first child of this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->

attribute: Notation.lastChild (inherited from Node)

Read-only  Yes
Type  Node
Description  The last child of this node. If there is no such node, this returns null.

Example

<!-- See example under Document interface -->
attribute: *Notation.previousSibling* (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately preceding this node. If there is no such node, this returns null.

**Example**

```xml
<!-- See example under Document interface -->
```

attribute: *Notation.nextSibling* (inherited from Node)

- **Read-only**: Yes
- **Type**: Node
- **Description**: The node immediately following this node. If there is no such node, this returns null.

**Example**

```xml
<!-- See example under Document interface -->
```

attribute: *Notation.attributes* (inherited from Node)

- **Read-only**: Yes
- **Type**: NamedNodeMap
- **Description**: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

**Example**

```xml
<!-- See example under Document interface -->
```
attribute: **Notation.ownerDocument**  
(inherited from Node)

- **Read-only**  Yes
- **Type**  Document
- **Description**  The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

**Example**

```html
<!-- See example under Document interface -->
```

**method: Notation.insertBefore(newChild, refChild)**  (inherited from Node)

- **Description**  Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.

- **Parameters**  
  - Node `newChild`—The node to insert. Node `refChild`—The reference node, i.e., the node before which the new node must be inserted.

- **Returns**  Node—The node being inserted.

**Exceptions**

- **DOMException HIERARCHY_REQUEST_ERR**  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node’s ancestors.

- **DOMException WRONG_DOCUMENT_ERR**  Raised if `newChild` was created from a different document than the one that created this node.

- **DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

- **DOMException NOT_FOUND_ERR**  Raised if `refChild` is not a child of this node.
If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

Example

<!-- See example under Document interface -->

**method: Notation.replaceChild(newChild, oldChild) (inherited from Node)**

**Description**  
Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed.

**Parameters**  
Node `newChild`—The new node to put in the child list.  
Node `oldChild`—The node being replaced in the list.

**Returns**  
Node—The node replaced.

**Exceptions**

**DOMException HIERARCHY_REQUEST_ERR**  
Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR**  
Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**  
Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  
Raised if `oldChild` is not a child of this node.

Example

<!-- See example under Document interface -->
method: `Notation.removeChild(oldChild)` (inherited from `Node`)

Description: Removes the child node indicated by `oldChild` from the list of children and returns it.

Parameters: `Node oldChild`—The node being removed.

Returns: `Node`—The node removed.

Exceptions

- `DOMException NO_MODIFICATION_ALLOWED_ERR` Raised if this node is read-only.
- `DOMException NOT_FOUND_ERR` Raised if `oldChild` is not a child of this node.

Example:

<!-- See example under Document interface -->

method: `Notation.appendChild(newChild)` (inherited from `Node`)

Description: Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

Parameters: `Node newChild`—The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

Returns: `Node`—The node added.

Exceptions

- `DOMException HIERARCHY_REQUEST_ERR` Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.
DOM Exception WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOM Exception NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example
<!-- See example under Document interface -->

**method: Notation.hasMoreChildren()**  (inherited from Node)

Description This is a convenient method to allow easy determination of whether a node has any children.

Parameters None.

Returns Boolean—True if the node has any children, false if the node has no children.

Exceptions None.

Example
<!-- See example under Document interface -->

**method: Notation.cloneNode(deep)**  (inherited from Node)

Description Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null).

Parameters Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

Returns Node—The duplicate node.

Exceptions None.
NOTE
Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example

<!-- See example under Document interface -->

PROCESSINGINSTRUCTION

Type of interface  Extended

Own properties

Attributes  target, data

Methods  None.

Inherited properties

Attributes  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods  insertBefore, replaceChild, removeChild, appendChild, hasChildNodes, cloneNode

Description  All “processing instructions” (i.e. <? ... ?>) from XML document are in DOM accessible through this interface.

attribute: ProcessingInstruction.target

Read-only  Yes

Type  DOMString

Description  The target of this processing instruction. XML defines this as being the first token following the markup that begins the processing instruction.
Example

```javascript
var pi = document.createProcessingInstruction('php',
  'echo("another example")');
var output = pi.target;
```

attribute: `ProcessingInstruction.data`

Read-only: No

Type: DOMString

Description: The content of this processing instruction. This is from the first non–white space character after the target to the character immediately preceding the `?>`.

Exception

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

Example

```javascript
var pi = document.createProcessingInstruction('php',
  'echo("another example")');
var output = pi.data;
```

attribute: `ProcessingInstruction.nodeName` (inherited from Node)

Read-only: Yes

Type: DOMString

Description: The name of this node, depending on its type.

Example

<!-- See example under Document interface -->

attribute: `ProcessingInstruction.nodeValue` (inherited from Node)

Read-only: No
Type   DOMString

Description   The value of this node, depending on its type.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR   This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

<!-- See example under Document interface -->

attribute: ProcessingInstruction.nodeType (inherited from Node)

Read-only   Yes
Type   unsigned short
Description   A code representing the type of the underlying object.

Example

<!-- See example under Document interface -->

attribute: ProcessingInstruction.parentNode (inherited from Node)

Read-only   Yes
Type   Node
Description   The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.
Example
<!-- See example under Document interface -->

**attribute: **`ProcessingInstruction.childNodes` *(inherited from Node)*

- **Read-only**  Yes
- **Type**  NodeList
- **Description**  A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

**NOTE**
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName() method.

Example
<!-- See example under Document interface -->

**attribute: **`ProcessingInstruction.firstChild` *(inherited from Node)*

- **Read-only**  Yes
- **Type**  Node
- **Description**  The first child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->
attribute: ProcessingInstruction
   .lastChild (inherited from Node)

   Read-only    Yes
   Type         Node
   Description  The last child of this node. If there is no such node, this returns null.

Example
   <!-- See example under Document interface -->

attribute: ProcessingInstruction
   .previousSibling (inherited from Node)

   Read-only    Yes
   Type         Node
   Description  The node immediately preceding this node. If there is no such node, this returns null.

Example
   <!-- See example under Document interface -->

attribute: ProcessingInstruction
   .nextSibling (inherited from Node)

   Read-only    Yes
   Type         Node
   Description  The node immediately following this node. If there is no such node, this returns null.

Example
   <!-- See example under Document interface -->
attribute: **ProcessingInstruction**

*attributes* (inherited from Node)

- **Read-only**: Yes
- **Type**: NamedNodeMap
- **Description**: A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

**Example**

<!-- See example under Document interface -->

attribute: **ProcessingInstruction**

*ownerDocument* (inherited from Node)

- **Read-only**: Yes
- **Type**: Document
- **Description**: The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

**Example**

<!-- See example under Document interface -->

**method**: **ProcessingInstruction.insertBefore**(newChild, refChild) (inherited from Node)

- **Description**: Inserts the node newChild before the existing child node refChild. If refChild is null, insert newChild at the end of the list of children.
- **Parameters**: Node newChild—The node to insert. Node refChild—The reference node, i.e., the node before which the new node must be inserted.
- **Returns**: Node—The node being inserted.
Exceptions

**DOMException HIERARCHY_REQUEST_ERR**  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR**  Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  Raised if `refChild` is not a child of this node.

**NOTE**

If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

**Example**

`<!-- See example under Document interface -->`

**method:** `ProcessingInstruction.replaceChild(newChild, oldChild)`  
(inherited from `Node`)

**Description**  Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If `newChild` is already in the tree, it is first removed.

**Parameters**  Node `newChild`—The new node to put in the child list. Node `oldChild`—The node being replaced in the list.

**Returns**  Node—The node replaced.
Exceptions

DOMException HIERARCHY_REQUEST_ERR  Raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node’s ancestors.

DOMException WRONG_DOCUMENT_ERR  Raised if newChild was created from a different document than the one that created this node.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

<!-- See example under Document interface -->

method: ProcessingInstruction.removeChild(oldChild) (inherited from Node)

Description  Removes the child node indicated by oldChild from the list of children and returns it.

Parameters  Node oldChild—The node being removed.

Returns  Node—The node removed.

Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

DOMException NOT_FOUND_ERR  Raised if oldChild is not a child of this node.

Example

<!-- See example under Document interface -->
method: **ProcessingInstruction.appendChild(newChild)** (inherited from Node)

**Description**  
Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

**Parameters**  
Node `newChild`—The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

**Returns**  
Node—The node added.

**Exceptions**

DOMException **HIERARCHY_REQUEST_ERR**  
Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.

DOMException **WRONG_DOCUMENT_ERR**  
Raised if `newChild` was created from a different document than the one that created this node.

DOMException **NO_MODIFICATION_ALLOWED_ERR**  
Raised if this node is read-only.

**Example**

```
<!-- See example under Document interface -->
```

method: **ProcessingInstruction.hasChildNodes()** (inherited from Node)

**Description**  
This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  
None.

**Returns**  
Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  
None.
method: `ProcessingInstruction.cloneNode(deep)` (inherited from Node)

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns null).

**Parameters**  Boolean deep—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

NOTE

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example

<!-- See example under Document interface -->

**TEXT**

**Type of interface**  Fundamental

**Own properties**

- **Attributes**  None.
- **Methods**  `splitText`
Inherited properties

Attributes  data, length, nodeName, nodeValue, 
nodetype, parentNode, childNodes, firstChild, 
lastChild, previousSibling, nextSibling, attributes, ownerDocument

Methods  substringData, appendData, insertData, 
deleteData, replaceData, insertBefore, replaceChild, 
removeChild, appendChild, hasChildNodes, cloneNode

Description  The Text interface represents the textual content 
termed character data in XML) of an Element or Attribute. If 
there is no markup inside an element’s content, the text is con-
tained in a single object implementing the Text interface that is 
the only child of the element. If there is markup, it is parsed into 
a list of elements and Text nodes that form the list of children of 
the element.

NOTE  When a document is first made available via the DOM, there is only one Text 
node for each block of text. Users may create adjacent Text nodes that repre-
sent the contents of a given element without any intervening markup, but 
should be aware that there is no way to represent the separations between 
these nodes in XML or HTML, so they will not (in general) persist between DOM 
editing sessions. The normalize() method on Element merges any such adja-
cent Text objects into a single node for each block of text; this is recommended 
before employing operations that depend on a particular document structure, 
such as navigation with XPointers.

attribute: Text.data (inherited from CharacterData)

Read-only  No

Type  DOMString

Description  The character data of the node that implements 
this interface.
Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR
This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example

<!-- See example under CharacterData interface -->

attribute: Text.length (inherited from CharacterData)

Read-only Yes
Type unsigned long
Description The number of 16-bit units that are available through data and the substringData() method.

Example

<!-- See example under CharacterData interface -->

attribute: Text.nodeName (inherited from Node)

Read-only Yes
Type DOMString
Description The name of this node, depending on its type.

attribute: TextnodeValue (inherited from Node)

Read-only No
Type DOMString
Description The value of this node, depending on its type.
Exceptions

DOMException NO_MODIFICATION_ALLOWED_ERR
This exception raises on setting when the node is read-only.

DOMException DOMSTRING_SIZE_ERR  This exception raises on retrieval when it would return more characters than fit in a DOMString variable on the implementation platform.

Example
<!-- See example under Document interface -->

attribute: Text.nodeType (inherited from Node)

Read-only  Yes
Type  unsigned short
Description  A code representing the type of the underlying object.

Example
<!-- See example under Document interface -->

attribute: Text.parentNode (inherited from Node)

Read-only  Yes
Type  Node
Description  The parent of this node. All nodes, except Document, DocumentFragment, and Attr, may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is null.

Example
<!-- See example under Document interface -->
attribute: `Text.childNodes` (inherited from Node)

Read-only Yes
Type NodeList
Description A NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

NOTE
The content of the returned NodeList is “live” in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the `getElementsByTagName()` method.

Example
<!-- See example under Document interface -->

attribute: `Text.firstChild` (inherited from Node)

Read-only Yes
Type Node
Description The first child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: `Text.lastChild` (inherited from Node)

Read-only Yes
Type Node
Description  The last child of this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: Text.previousSibling (inherited from Node)

Read-only  Yes
Type  Node
Description  The node immediately preceding this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: Text.nextSibling (inherited from Node)

Read-only  Yes
Type  Node
Description  The node immediately following this node. If there is no such node, this returns null.

Example
<!-- See example under Document interface -->

attribute: Text.attributes (inherited from Node)

Read-only  Yes
Type  NamedNodeMap
Description  A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
attribute: Text.ownerDocument (inherited from Node)

Read-only  Yes
Type  Document

Description  The Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document, this is null.

Example
<!-- See example under Document interface -->

method: Text.splitText(offset)

Description  Breaks this Text node into two Text nodes at the specified offset, keeping both in the tree as siblings. This node then only contains all the content up to the offset point. And a new Text node, which is inserted as the next sibling of this node, contains all the content at and after the offset point.

Parameters  unsigned long offset—The 16-bit unit offset at which to split, starting from 0.

Returns  Text—The new Text node.

Exceptions

DOMException INDEX_SIZE_ERR  Raised if the specified offset is negative or greater than the number of 16-bit unit in data.

DOMException NO_MODIFICATION_ALLOWED_ERR  Raised if this node is read-only.

Example

var txt = document.createTextNode('AAA-BBB-CCC');
var output1 = txt.splitText(4).data;
var output2 = txt.data;
### method: `Text.substringData(offset, count)` (inherited from CharacterData)

**Description**  
Extracts a range of data from the node.

**Parameters**  
- `offset`: Start offset of substring to extract.  
- `count`: The number of 16-bit units to extract.

**Returns**  
`DOMString`—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.

**Exceptions**
- `DOMException INDEX_SIZE_ERR`  
  Raised if the specified offset is negative or greater than the number of 16-bit units in data, or if the specified count is negative.
- `DOMException DOMSTRING_SIZE_ERR`  
  Raised if the specified range of text does not fit into a `DOMString`.

**Example**

<!-- See example under CharacterData interface -->

### method: `Text.appendData(arg)` (inherited from CharacterData)

**Description**  
Appends the string to the end of the character data of the node.

**Parameters**  
- `arg`: The `DOMString` to append.

**Returns**  
Nothing.

**Exceptions**
- `DOMException NO_MODIFICATION_ALLOWED_ERR`  
  Raised if this node is read-only.

**Example**

<!-- See example under CharacterData interface -->
method: `Text.insertData(offset, arg)` (inherited from CharacterData)

**Description**
Inserts a string at the specified 16-bit units offset.

**Parameters**
- `offset`: unsigned long offset—The 16-bit units offset at which to insert.
- `arg`: DOMString—The DOMString to insert.

**Returns**
Nothing.

**Exceptions**
- **DOMException INDEX_SIZE_ERR**
  Raised if the specified offset is negative or greater than the number of 16-bit units in data.
- **DOMException NO_MODIFICATION_ALLOWED_ERR**
  Raised if this node is read-only.

Example
<!-- See example under CharacterData interface -->

**method: `Text.deleteData(offset, count)`** (inherited from CharacterData)

**Description**
Removes a range of 16-bit units from the node. Upon success, data and length reflect the change.

**Parameters**
- `offset`: unsigned long offset—The offset from which to remove characters.
- `count`: unsigned long count—The number of 16-bit units to delete. If the sum of offset and count exceeds length, then all 16-bit units from offset to the end of the data are deleted.

**Returns**
Nothing.

**Exceptions**
- **DOMException INDEX_SIZE_ERR**
  Raised if the specified offset is negative or greater than the number of characters in data, or if the specified count is negative.
- **DOMException NO_MODIFICATION_ALLOWED_ERR**
  Raised if this node is read-only.
Example

<!-- See example under CharacterData interface -->

method: `Text.replaceData(offset, count, arg)` (inherited from CharacterData)

**Description**  
Replaces the characters starting at the specified 16-bit units offset with the specified string.

**Parameters**  
- `unsigned long offset`: The offset from which to start replacing.  
- `unsigned long count`: The number of 16-bit units to replace. If the sum of offset and count exceeds `length`, then all 16-bit units to the end of the data are replaced.  
- `DOMString arg`: The `DOMString` with which the range must be replaced.

**Returns**  
Nothing.

**Exceptions**

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

- `DOMException NO_MODIFICATION_ALLOWED_ERR`: Raised if this node is read-only.

Example

<!-- See example under CharacterData interface -->

method: `Text.insertBefore(newChild, refChild)` (inherited from Node)

**Description**  
Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is null, inserts `newChild` at the end of the list of children.

**Parameters**  
- `Node newChild`: The node to insert.  
- `Node refChild`: The reference node, i.e., the node before which the new node must be inserted.

**Returns**  
`Node`: The node being inserted.
Exceptions

**DOMException HIERARCHY_REQUEST_ERR** Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR** Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR** Raised if this node is read-only.

**DOMException NOT_FOUND_ERR** Raised if `refChild` is not a child of this node.

**NOTE**

If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If `newChild` is already in the tree, it is first removed.

Example

`<!-- See example under Document interface -->`

**method**: `Text.replaceChild(newChild, oldChild)` *(inherited from Node)*

**Description** Replaces the child node `oldChild` with `newChild` in the list of children and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed.

**Parameters**
- Node `newChild`—The new node to put in the child list.
- Node `oldChild`—The node being replaced in the list.

**Returns** Node—The node replaced.
Exceptions

**DOMException HIERARCHY_REQUEST_ERR**  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node’s ancestors.

**DOMException WRONG_DOCUMENT_ERR**  Raised if `newChild` was created from a different document than the one that created this node.

**DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  Raised if `oldChild` is not a child of this node.

Example

<!-- See example under Document interface -->

**method: Text.removeChild(oldChild)**  (inherited from Node)

**Description**  Removes the child node indicated by `oldChild` from the list of children and returns it.

**Parameters**  Node `oldChild`—The node being removed.

**Returns**  Node—The node removed.

Exceptions

**DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**DOMException NOT_FOUND_ERR**  Raised if `oldChild` is not a child of this node.

Example

<!-- See example under Document interface -->
method: `Text.appendChild(newChild)` (inherited from Node)

**Description**  Adds the node `newChild` to the end of the list of children of this node. If `newChild` is already in the tree, it is first removed.

**Parameters**  Node `newChild`—The node to add. If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node.

**Returns**  Node—The node added.

**Exceptions**

- **DOMException HIERARCHY_REQUEST_ERR**  Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node’s ancestors.

- **DOMException WRONG_DOCUMENT_ERR**  Raised if `newChild` was created from a different document than the one that created this node.

- **DOMException NO_MODIFICATION_ALLOWED_ERR**  Raised if this node is read-only.

**Example**

```xml
<!-- See example under Document interface -->
```

method: `Text.hasChildNodes()` (inherited from Node)

**Description**  This is a convenient method to allow easy determination of whether a node has any children.

**Parameters**  None.

**Returns**  Boolean—True if the node has any children, false if the node has no children.

**Exceptions**  None.
Example

`<!-- See example under Document interface -->`

**method: Text.cloneNode(deep) (inherited from Node)**

**Description**  Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns null).

**Parameters**  Boolean `deep`—If true, recursively clone the sub-tree under the specified node; if false, clone only the node itself (and its attributes, if it is an Element).

**Returns**  Node—The duplicate node.

**Exceptions**  None.

---

**NOTE**

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Example

`<!-- See example under Document interface -->`