13 - OCT - 2008

it has only 7 methods

5 - methods for attribute node itself
1 - get parent
1 - get schema type

17 - Dec - 08

org.w3c.dom

Interface Attr

it has ONLY Text node as child element. (i tested)

All Superinterfaces:

Node

there is no Setter method for name of attribute.

Super ... great design. if not, it will give collapse, since this is object, point to an attribute of an element. if i change name, then actual name will be changed and in consistent with actual xml document which was parsed

```
public interface Attr
```

extends Node

The `Attr` interface represents an attribute in an `Element` object. Typically the allowable values for the attribute are defined in a schema associated with the document.

`Attr` objects inherit the `Node` interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the `Node` attributes `parentNode`, `previousSibling`, and `nextSibling` have a `null` value for `Attr` objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with; this should make it more efficient to implement such features as default attributes associated with all elements of a given type. Furthermore, `Attr` nodes may not be immediate children of a `DocumentFragment`. However, they can be associated with `Element` nodes contained within a `DocumentFragment`. In short, users and implementors of the DOM need to be aware that `Attr` nodes have some things in common with other objects inheriting the `Node` interface, but they also are quite distinct.

The attribute's effective value is determined as follows: if this attribute has been explicitly assigned any value, that value is the attribute's effective value; otherwise, if there is a declaration for this attribute, and that declaration includes a default value, then that default value is the attribute's effective value; otherwise, the attribute does not exist on this element in the structure model until it has been explicitly added. Note that the `Node.nodeValue` attribute on the `Attr` instance can also be used to retrieve the string version of the attribute's value(s).

so, default value can be associated with attribute only by schema

If the attribute was not explicitly given a value in the instance document but has a default value provided by the schema associated with the document, an attribute node will be created with `specified` set to `false`. Removing attribute nodes for which a default value is defined in the schema generates a new attribute node with the default value and `specified` set to `false`. If validation occurred while invoking `Document.normalizeDocument()`, attribute nodes with `specified` equals to `false` are recomputed according to the default attribute values provided by the schema. If no default value is associate with this attribute in the schema, the attribute node is discarded.

In XML, where the value of an attribute can contain entity references, the child nodes of the `Attr` node may be either `Text` or `EntityReference` nodes (when these are in use; see the description of `EntityReference` for discussion).

The DOM Core represents all attribute values as simple strings, even if the DTD or schema associated with the document

declares them of some specific type such as tokenized.

The way attribute value normalization is performed by the DOM implementation depends on how much the implementation knows about the schema in use. Typically, the `value` and `nodeValue` attributes of an `Attr` node initially returns the normalized value given by the parser. It is also the case after `Document.normalizeDocument()` is called (assuming the right options have been set). But this may not be the case after mutation, independently of whether the mutation is performed by setting the string value directly or by changing the `Attr` child nodes. In particular, this is true when character references are involved, given that they are not represented in the DOM and they impact attribute value normalization. On the other hand, if the implementation knows about the schema in use when the attribute value is changed, and it is of a different type than CDATA, it may normalize it again at that time. This is especially true of specialized DOM implementations, such as SVG DOM implementations, which store attribute values in an internal form different from a string.

The following table gives some examples of the relations between the attribute value in the original document (parsed attribute), the value as exposed in the DOM, and the serialization of the value:

Examples	Parsed attribute value	Initial `Attr.value`	Serialized attribute value
Character reference	`"x²=5"`	`"x²=5"`	`"x²=5"`
Built-in character entity	`"y<6"`	`"y<6"`	`"y<6"`
Literal newline between	`"x=5
y=6"`	`"x=5 y=6"`	`"x=5
y=6"`
Normalized newline between	`"x=5 y=6"`	`"x=5 y=6"`	`"x=5 y=6"`
Entity `e` with literal newline	`<!ENTITY e '...
...'> [...]> "x=5&e;y=6"`	*Dependent on Implementation and Load Options*	*Dependent on Implementation and Load/Save Options*

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

there is no Setter method for name of attribute.

Super ... great design. if not, it will give collapse, since this is object, point to an attribute of an element. if i change name, then actual name will be changed and in consistent with actual xml document which was parsed

java.lang.String	**getName**() Returns the name of t
Element	**getOwnerElement**() The Element node this attribute is attached to or null if this attribute is not in use.
TypeInfo	**getSchemaTypeInfo**() The type information associated with this attribute.
boolean	**getSpecified**() True if this attribute was explicitly given a value in the instance document, false otherwise.
java.lang.String	**getValue**() On retrieval, the value of the attribute is returned as a string.
boolean	**isId**() Returns whether this attribute is known to be of type ID (i.e. to contain an identifier for its owner element) or not.
void	**setValue**(java.lang.String value) On retrieval, the value of the attribute is returned as a string.

wow

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

Method Detail

getName

`java.lang.String` **`getName`**`()`

Returns the name of this attribute. If `Node.localName` is different from `null`, this attribute is a qualified name.

---

getSpecified

`boolean` **`getSpecified`**`()`

`True` if this attribute was explicitly given a value in the instance document, `false` otherwise. If the application changed the value of this attribute node (even if it ends up having the same value as the default value) then it is set to `true`. The implementation may handle attributes with default values from other schemas similarly but applications should use `Document.normalizeDocument()` to guarantee this information is up-to-date.

---

getValue

`java.lang.String` **`getValue`**`()`

On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. See also the method `getAttribute` on the `Element` interface.
On setting, this creates a `Text` node with the unparsed contents of the string, i.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `Element.setAttribute()`.
Some specialized implementations, such as some [SVG 1.1] implementations, may do normalization automatically, even after mutation; in such case, the value on retrieval may differ from the value on setting.

---

setValue

value can have space

`void` **`setValue`**`(java.lang.String value)`
`throws` `DOMException`

On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. See also the method `getAttribute` on the `Element` interface.
On setting, this creates a `Text` node with the unparsed contents of the string, i.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `Element.setAttribute()`.
Some specialized implementations, such as some [SVG 1.1] implementations, may do normalization automatically, even after mutation; in such case, the value on retrieval may differ from the value on setting.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

---

getOwnerElement

`Element getOwnerElement()`

The `Element` node this attribute is attached to or `null` if this attribute is not in use.

Since:
DOM Level 2

---

getSchemaTypeInfo

`TypeInfo getSchemaTypeInfo()`

The type information associated with this attribute. While the type information contained in this attribute is guarantee to be correct after loading the document or invoking `Document.normalizeDocument()`, `schemaTypeInfo` may not be reliable if the node was moved.

Since:
DOM Level 3

---

isId

`boolean isId()`

Returns wheth ... lement) or not.
When it is an ... method
`Document.` ... attribute node is
known to con ...

this is only way to say ID programatically.. No.. no..

Element has method to set any ARBITRARY attribute can be ID

(i tested)

```
attribute = element.getAttributeNode("name");
element.setIdAttributeNode(attribute, true);
logger.info("id attribute "+attribute.isId());
```

- If valid ... ment or while invoki ... ributions (PSVI contrib ... ute using the schem ...
- If validation occurred using a DTD while loading the document or while invoking `Document.normalizeDocument()`, the infoset **[type definition]** value is used to determine if this attribute is a DTD-determined ID attribute using the DTD-determined ID definition in [XPointer] .
- from the use of the methods `Element.setIdAttribute()`, `Element.setIdAttributeNS()`, or `Element.setIdAttributeNode()`, i.e. it is an user-determined ID attribute;

  Note: XPointer framework (see section 3.2 in [XPointer]) consider the DOM user-determined ID attribute

as being part of the XPointer externally-determined ID definition.
- using mechanisms that are outside the scope of this specification, it is then an externally-determined ID attribute. This includes using schema languages different from XML schema and DTD.

If validation occurred while invoking `Document.normalizeDocument()`, all user-determined ID attributes are reset and all attribute nodes ID information are then reevaluated in accordance to the schema used. As a consequence, if the `Attr.schemaTypeInfo` attribute contains an ID type, `isId` will always return true.

Since:

DOM Level 3

---

14 - OCT - 2008

18 - Dec - 2008

so, CDATAsection object can be only with XML

org.w3c.dom

Interface CDATASection

Haa ... Haa.. No methods in this interface

All Superinterfaces:

CharacterData, Node, Text

CDATASection object CANNOT associated with Document object. it can be only with Element. even it cannot be with Attr. (i tested)

```
public interface CDATASection
```

extends Text

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections cannot be nested. Their primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The `CharacterData.data` attribute holds the text that is contained by the CDATA section. Note that this *may* contain characters that need to be escaped outside of CDATA sections and that, depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The `CDATASection` interface inherits from the `CharacterData` interface through the `Text` interface. Adjacent `CDATASection` nodes are not merged by use of the `normalize` method of the `Node` interface.

No lexical check is done on the content of a CDATA section and it is therefore possible to have the character sequence `"]]>"` in the content, which is illegal in a CDATA section per section 2.7 of [XML 1.0]. The presence of this character sequence must generate a fatal error during serialization or the cdata section must be splitted before the serialization (see also the parameter `"split-cdata-sections"` in the `DOMConfiguration` interface).

Note: Because no markup is recognized within a `CDATASection`, character numeric references cannot be used as an escape mechanism when serializing. Therefore, action needs to be taken when serializing a

CDATASection with a character encoding where some of the contained characters cannot be represented. Failure to do so would not produce well-formed XML.

Note: One potential solution in the serialization process is to end the CDATA section before the character, output the character using a character reference or entity reference, and open a new CDATA section for any further characters in the text node. Note, however, that some code conversion libraries at the time of writing do not return an error or exception when a character is missing from the encoding, making the task of ensuring that data is not corrupted on serialization more difficult.

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

Methods inherited from interface org.w3c.dom.Text

getWholeText, isElementContentWhitespace, replaceWholeText, splitText

Methods inherited from interface org.w3c.dom.CharacterData

appendData, deleteData, getData, getLength, insertData, replaceData, setData, substringData

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

13 - OCT - 2008 18 - Dec - 08

org.w3c.dom

Interface CharacterData

only 8 methods
All of methods are self related. no Child, No parent

All Superinterfaces:

Node

All Known Subinterfaces:

CDATASection, Comment, Text

So, CharacterData is used indirectly only.

`public interface CharacterData`

```
private void evaluateCharacterData(Element element) {

  Element b = (Element) (element.getElementsByTagName("b")).item(0);
  CharacterData characterData = (CharacterData)b.getFirstChild();
  selfMethods(characterData);
```

extends Node

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`.

As explained in the `DOMString` interface, text strings in the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In the following, the term 16-bit units is used whenever necessary to indicate that indexing on CharacterData is done in 16-bit units.

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

void	**appendData**(java.lang.String arg) Append the string to the end of the character data of the node.
void	**deleteData**(int offset, int count) Remove a range of 16-bit units from the node.
java.lang.String	**getData**() The character data of the node that implements this interface.
int	**getLength**() The number of 16-bit units that are available through `data` and the `substringData` method below.
void	**insertData**(int offset, java.lang.String arg) Insert a string at the specified 16-bit unit offset.
void	**replaceData**(int offset, int count, java.lang.String arg) Replace the characters starting at the specified 16-bit unit offset with the specified string.
void	**setData**(java.lang.String data) The character data of the node that implements this interface.
java.lang.String	**substringData**(int offset, int count) Extracts a range of data from the node.

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

Method Detail

getData

```
java.lang.String getData()
                         throws DOMException
```

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a single `DOMString`. In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

that is , a elment data data will be copied into DOMString then return this domstring object

Throws:

DOMException - DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

---

setData

```
void setData(java.lang.String data)
             throws DOMException
```

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a

single DOMString. In such cases, the user may call substringData to retrieve the data in appropriately sized pieces.

Throws:

DOMException - NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

---

getLength

```
int getLength()
```

The number of 16-bit units that are available through data and the substringData method below. This may have the value zero, i.e., CharacterData nodes may be empty.

---

substringData

```
java.lang.String substringData(int offset,
                               int count)
                               throws DOMExceptio
```

i can give any value, which is not really available in data

this is the bug in Implementation, they have to raise exception like "OutofIndexExcepti on"

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:

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.

Throws:

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

---

appendData

```
void appendData(java.lang.String arg)
                throws DOMException
```

Append the string to the end of the character data of the node. Upon success, `data` provides access to the concatenation of `data` and the `DOMString` specified.

Parameters:

`arg` - The `DOMString` to append.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

---

insertData

```
void insertData(int offset,
                java.lang.String arg)
                throws DOMException
```

Insert a string at the specified 16-bit unit offset.

Parameters:

`offset` - The character offset at which to insert.

`arg` - The `DOMString` to insert.

Throws:

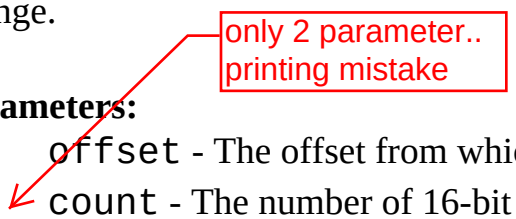
`DOMException` - INDEX_SIZE_ERR: Raised if the specified `offset` is negative or greater than the number of 16-bit units in `data`.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

---

deleteData

```
void deleteData(int offset,
                int count)
                throws DOMException
```

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



Parameters:

`offset` - The offset from which to start removing.

`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.

Throws:

`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.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

---

replaceData

```
void replaceData(int offset,
                 int count,
                 java.lang.String arg)
                 throws DOMException
```

Replace the characters starting at the specified 16-bit unit 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; (i.e., the effect is the same as a `remove` method call with the same range, followed by an `append` method invocation).

`arg` - The `DOMString` with which the range must be replaced.

Throws:

`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.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

---

14 - OCT - 2008

18 - Dec - 2008

no methods :)

org.w3c.dom

Interface Comment

All Superinterfaces:

CharacterData, Node

`public interface` **`Comment`**

extends CharacterData

This interface inherits from `CharacterData` and represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'. ~~Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.~~

~~No lexical check is done on the content of a comment and it is therefore possible to have the character sequence "`--`" (double-hyphen) in the content, which is illegal in a comment per section 2.5 of [XML 1.0]. The presence of this character sequence must generate a fatal error during serialization.~~

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

Methods inherited from interface org.w3c.dom.CharacterData

appendData, deleteData, getData, getLength, insertData, replaceData, setData, substringData

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

http://www-labs.det.uvigo.es/documentation/LRO/jaxp/jaxp-1_3-html/index.html

for examples for DOM
http://www.java-tips.org/org.w3c.dom/

example for some area
http://www.cafeconleche.org/books/xmljava/chapters/index.html

Java API for XML Processing (JAXP) 1.4

JAXP has SIX Major parts
1. SAX, 2. DOM, 3. TrAX, 4. XPath, 5. Validation, 6. StAX

Packages	
javax.xml	Defines core XML constants and functionality from the XML specifications.
javax.xml.datatype	XML/Java Type Mappings.
javax.xml.namespace	XML Namespace processing.
javax.xml.parsers	Provides classes allowing the processing of XML documents.
javax.xml.stream	
javax.xml.stream.events	
javax.xml.stream.util	
javax.xml.transform	This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.
javax.xml.transform.dom	This package implements DOM-specific transformation APIs.
javax.xml.transform.sax	This package implements SAX2-specific transformation APIs.
javax.xml.transform.stax	Provides for StAX-specific transformation APIs.
javax.xml.transform.stream	This package implements stream- and URI- specific transformation APIs.
javax.xml.validation	This package provides an API for validation of XML documents.

since JAXP 1.3

some SAX example
http://www.java-tips.org/java-se-tips/org.xml.sax/

next round, please set the factory SYSTEM property in every example

it is Introduced by SUN

1st round - SUN Material
2nd round - Deleted some of page
3rd round - deleted sun material itself
finished on 22 - Nov - 08
4th round - 24 Nov 2008

1
2
3
4
5

11
3
5
2
2

javax.xml.xpath	This package provides an *object-model neutral* API for the evaluation of XPath expressions and access to the evaluation environment.
org.w3c.dom	
org.w3c.dom.bootstrap	
org.w3c.dom.events	
org.w3c.dom.ls	
org.w3c.dom.ranges	
org.w3c.dom.traversal	
org.xml.sax	This package provides the core SAX APIs.
org.xml.sax.ext	This package contains interfaces to SAX2 facilities that conformant SAX drivers won't necessarily support.
org.xml.sax.helpers	This package contains "helper" classes, including support for bootstrapping SAX-based applications.

for examples for DOM
http://www.java-tips.org/org.w3c.dom/

5 -packages

DOM 3

DOM 2

DOM 2

DOM

SAX 3 - Packages

At 5th round get familier with DOMConfiguration object

15 - OCT - 2008

16 - Dec - 08

org.w3c.dom

Interface Document

All Superinterfaces:

Node

See my Excel for complete method grouping

THIS CLASSS HAS 26 METHODS

8 factory methods
7 - methods for document itself
5 - generic method
2 - getter / setter for location
4 - getter for child

```
public interface Document
```

extends Node

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 a `ownerDocument` attribute which associates them with the `Document` within whose context they were created.~~

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

`ATTRIBUTE_NODE`, `CDATA_SECTION_NODE`, `COMMENT_NODE`, `DOCUMENT_FRAGMENT_NODE`, `DOCUMENT_NODE`, `DOCUMENT_POSITION_CONTAINED_BY`, `DOCUMENT_POSITION_CONTAINS`, `DOCUMENT_POSITION_DISCONNECTED`, `DOCUMENT_POSITION_FOLLOWING`, `DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC`, `DOCUMENT_POSITION_PRECEDING`, `DOCUMENT_TYPE_NODE`, `ELEMENT_NODE`, `ENTITY_NODE`, `ENTITY_REFERENCE_NODE`, `NOTATION_NODE`, `PROCESSING_INSTRUCTION_NODE`, `TEXT_NODE`

Method Summary

Node	**adoptNode**(Node source) Attempts to adopt a node from another document to this document.
Attr	**createAttribute**(java.lang.String name) Creates an Attr of the given name.
Attr	**createAttributeNS**(java.lang.String namespaceURI, java.lang.String qualifiedName) Creates an attribute of the given qualified name and namespace URI.
CDATASection	**createCDATASection**(java.lang.String data) Creates a CDATASection node whose value is the specified string.
Comment	**createComment**(java.lang.String data) Creates a Comment node given the specified string.
DocumentFragment	**createDocumentFragment**() Creates an empty DocumentFragment object.
Element	**createElement**(java.lang.String tagName) Creates an element of the type specified.
Element	**createElementNS**(java.lang.String namespaceURI, java.lang.String qualifiedName) Creates an element of the given qualified name and namespace URI.
EntityReference	**createEntityReference**(java.lang.String name) Creates an EntityReference object.
ProcessingInstruction	**createProcessingInstruction**(java.lang.String target, java.lang.String data) Creates a ProcessingInstruction node given the specified name and data strings.
Text	**createTextNode**(java.lang.String data) Creates a Text node given the specified string.
DocumentType	**getDoctype**() The Document Type Declaration (see DocumentType) associated with this document.
Element	**getDocumentElement**() This is a convenience attribute that allows direct access to the child node that is the document element of the document.

it is equal to namespace declaration

they could have renamed like getDocumentRoot Element()

java.lang.String	**getDocumentURI**() The location of the document or null if undefined or if the Document was created using DOMImplementation.createDocument.
DOMConfiguration	**getDomConfig**() The configuration used when Document.normalizeDocument() is invoked.
Element	**getElementById**(java.lang.String elementId) Returns the Element that has an ID attribute with the given value.
NodeList	**getElementsByTagName**(java.lang.String tagname) Returns a NodeList of all the Elements in document order with a given tag name and are contained in the document.
NodeList	**getElementsByTagNameNS**(java.lang.String namespaceURI, java.lang.String localName) Returns a NodeList of all the Elements with a given local name and namespace URI in document order.
DOMImplementation	**getImplementation**() The DOMImplementation object that handles this document.
java.lang.String	**getInputEncoding**() An attribute specifying the encoding used for this document at the time of the parsing.
boolean	**getStrictErrorChecking**() An attribute specifying whether error checking is enforced or not.
java.lang.String	**getXmlEncoding**() An attribute specifying, as part of the XML declaration, the encoding of this document.
boolean	**getXmlStandalone**() An attribute specifying, as part of the XML declaration, whether this document is standalone.
java.lang.String	**getXmlVersion**() An attribute specifying, as part of the XML declaration, the version number of this document.

one localName parameter method

Node	**importNode**(Node importedNode, boolean deep) Imports a node from another document to this document, without altering or removing the source node from the original document; this method creates a new copy of the source node.
void	**normalizeDocument**() This method acts as if the document was going through a save and load cycle, putting the document in a "normal" form.
Node	**renameNode**(Node n, java.lang.String namespaceURI, java.lang.String qualifiedName) Rename an existing node of type ELEMENT_NODE or ATTRIBUTE_NODE.
void	**setDocumentURI**(java.lang.String documentURI) The location of the document or null if undefined or if the Document was created using DOMImplementation.createDocument.
void	**setStrictErrorChecking**(boolean strictErrorChecking) An attribute specifying whether error checking is enforced or not.
void	**setXmlStandalone**(boolean xmlStandalone) An attribute specifying, as part of the XML declaration, whether this document is standalone.
void	**setXmlVersion**(java.lang.String xmlVersion) An attribute specifying, as part of the XML declaration, the version number of this document.

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

Method Detail

getDoctype

`DocumentType` **`getDoctype()`**

The Document Type Declaration (see `DocumentType`) associated with this document. For XML documents without a document type declaration this returns `null`. For HTML documents, a `DocumentType` object may be returned, independently of the presence or absence of document type declaration in the HTML document.
This provides direct access to the `DocumentType` node, child node of this `Document`. This node can be set at document creation time and later changed through the use of child nodes manipulation methods, such as `Node.insertBefore`, or `Node.replaceChild`. Note, however, that while some implementations may instantiate different types of `Document` objects supporting additional features than the "Core", such as "HTML" [DOM Level 2 HTML], based on the `DocumentType` specified at creation time, changing it afterwards is very unlikely to result in a change of the features supported.

Since:
DOM Level 3

---

getImplementation

`DOMImplementation` **`getImplementation()`**

The `DOMImplementation` object that handles this document. A DOM application may use objects from multiple implementations.

---

getDocumentElement

it will retrieve Root Element

`Element` **`getDocumentElement()`**

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

---

createElement

`Element` **`createElement`**`(java.lang.String tagName)`
`throws` `DOMException`

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.
To create an element with a qualified name and namespace URI, use the `createElementNS` method.

Parameters:

`tagName` - The name of the element type to instantiate. For XML, this is case-sensitive, otherwise it depends on the case-sensitivity of the markup language in use. In that case, the name is mapped to the canonical form of that markup by the DOM implementation.

Returns:

A new `Element` object with the `nodeName` attribute set to `tagName`, and `localName`, `prefix`, and `namespaceURI` set to `null`.

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified name is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.

---

createDocumentFragment

`DocumentFragment` **`createDocumentFragment`**`()`

Creates an empty `DocumentFragment` object.

Returns:

A new `DocumentFragment`.

---

createTextNode

`Text` **`createTextNode`**`(java.lang.String data)`

Creates a `Text` node given the specified string.

Parameters:

`data` - The data for the node.

Returns:

The new `Text` object.

---

createComment

`Comment` **`createComment`**`(java.lang.String data)`

Creates a `Comment` node given the specified string.

Parameters:

`data` - The data for the node.

Returns:

The new `Comment` object.

---

createCDATASection

`CDATASection` **`createCDATASection`**`(java.lang.String data)`
`throws` `DOMException`

so, CDATAsection object can be only with XML

Creates a `CDATASection` node whose value is the specified string.

Parameters:

`data` - The data for the `CDATASection` contents.

Returns:

The new `CDATASection` object.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

---

createProcessingInstruction

`ProcessingInstruction` **`createProcessingInstruction`**`(java.lang.String target,`

java.lang.String data)
throws DOMException

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

Parameters:

target - The target part of the processing instruction.Unlike Document.createElementNS or Document.createAttributeNS, no namespace well-formed checking is done on the target name. Applications should invoke Document.normalizeDocument() with the parameter "namespaces" set to true in order to ensure that the target name is namespace well-formed.

data - The data for the node.

Returns:

The new ProcessingInstruction object.

Throws:

DOMException - INVALID_CHARACTER_ERR: Raised if the specified target is not an XML name according to the XML version in use specified in the Document.xmlVersion attribute.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

---

createAttribute

Attr **createAttribute**(java.lang.String name)
throws DOMException

Creates an Attr of the given name. Note that the Attr instance can then be set on an Element using the setAttributeNode method.

To create an attribute with a qualified name and namespace URI, use the createAttributeNS method.

Parameters:

name - The name of the attribute.

Returns:

A new Attr object with the nodeName attribute set to name, and localName, prefix, and namespaceURI set to null. The value of the attribute is the empty string.

Throws:

DOMException - INVALID_CHARACTER_ERR: Raised if the specified name is not an XML name according to the XML version in use specified in the Document.xmlVersion attribute.

---

createEntityReference

`EntityReference` **`createEntityReference`**`(java.lang.String name)`
`throws DOMException`

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.

Note: If any descendant of the `Entity` node has an unbound namespace prefix, the corresponding descendant of the created `EntityReference` node is also unbound; (its `namespaceURI` is `null`). The DOM Level 2 and 3 do not support any mechanism to resolve namespace prefixes in this case.

Parameters:

`name` - The name of the entity to reference.Unlike `Document.createElementNS` or `Document.createAttributeNS`, no namespace well-formed checking is done on the entity name. Applications should invoke `Document.normalizeDocument()` with the parameter "namespaces" set to `true` in order to ensure that the entity name is namespace well-formed.

Returns:

The new `EntityReference` object.

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified name is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.
NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

---

getElementsByTagName

`NodeList` **`getElementsByTagName`**`(java.lang.String tagname)`

Returns a `NodeList` of all the `Elements` in document order with a given tag name and are contained in the document.

Yes, it is check "case-senstive"

Parameters:

`tagname` - The name of the tag to match on. The special value "*" matches all tags. For XML, the `tagname` parameter is case-sensitive, otherwise it depends on the case-sensitivity of the

markup language in use.

Returns:

A new `NodeList` object containing all the matched `Elements`.

---

importNode

```
Node importNode(Node importedNode,
                boolean deep)
                throws DOMException
```

Just compare with adoptNode() method of this class.

importNode -for COPY
adoptNode - for MOVE

Imports a node from another document to this document, without altering or removing the source node from the original document; this method creates a new copy of the source node. The returned node has no parent; (`parentNode` is `null`).

For all nodes, importing a node creates a node object owned by the importing document, with attribute values identical to the source node's `nodeName` and `nodeType`, plus the attributes related to namespaces (`prefix`, `localName`, and `namespaceURI`). As in the `cloneNode` operation, the source node is not altered. User data associated to the imported node is not carried over. However, if any `UserDataHandlers` has been specified along with the associated data these handlers will be called with the appropriate parameters before this method returns.

Additional information is copied as appropriate to the `nodeType`, attempting to mirror the behavior expected if a fragment of XML or HTML source was copied from one document to another, recognizing that the two documents may have different DTDs in the XML case. The following list describes the specifics for each type of node.

ATTRIBUTE_NODE

The `ownerElement` attribute is set to `null` and the `specified` flag is set to `true` on the generated `Attr`. The descendants of the source `Attr` are recursively imported and the resulting nodes reassembled to form the corresponding subtree. Note that the `deep` parameter has no effect on `Attr` nodes; they always carry their children with them when imported.

DOCUMENT_FRAGMENT_NODE

it will have only a Text Node as child

If the `deep` option was set to `true`, the descendants of the source `DocumentFragment` are recursively imported and the resulting nodes reassembled under the imported `DocumentFragment` to form the corresponding subtree. Otherwise, this simply generates an empty `DocumentFragment`.

DOCUMENT_NODE

`Document` nodes cannot be imported.

DOCUMENT_TYPE_NODE

`DocumentType` nodes cannot be imported.

ELEMENT_NODE

Specified attribute nodes of the source element are imported, and the generated `Attr` nodes are attached to the generated `Element`. Default attributes are *not* copied, though if the document

being imported into defines default attributes for this element name, those are assigned. If the `importNode` `deep` parameter was set to `true`, the descendants of the source element are recursively imported and the resulting nodes reassembled to form the corresponding subtree.

~~ENTITY_NODE~~

~~`Entity` nodes can be imported, however in the current release of the DOM the `DocumentType` is readonly. Ability to add these imported nodes to a `DocumentType` will be considered for addition to a future release of the DOM.On import, the `publicId`, `systemId`, and `notationName` attributes are copied. If a `deep` import is requested, the descendants of the the source `Entity` are recursively imported and the resulting nodes reassembled to form the corresponding subtree.~~

~~ENTITY_REFERENCE_NODE~~

~~Only the `EntityReference` itself is copied, even if a `deep` import is requested, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.~~

~~NOTATION_NODE~~

~~`Notation` nodes can be imported, however in the current release of the DOM the `DocumentType` is readonly. Ability to add these imported nodes to a `DocumentType` will be considered for addition to a future release of the DOM.On import, the `publicId` and `systemId` attributes are copied. Note that the `deep` parameter has no effect on this type of nodes since they cannot have any children.~~

these 4 nodes never have childre an

~~PROCESSING_INSTRUCTION_NODE~~

~~The imported node copies its `target` and `data` values from those of the source node.Note that the `deep` parameter has no effect on this type of nodes since they cannot have any children~~.

TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE

These three types of nodes inheriting from `CharacterData` copy their `data` and `length` attributes from those of the source node.Note that the `deep` parameter has no effect on these types of nodes since they cannot have any children.

Parameters:

`importedNode` - The node to import.

`deep` - If `true`, recursively import the subtree under the specified node; if `false`, import only the node itself, as explained above. This has no effect on nodes that cannot have any children, and on `Attr`, and `EntityReference` nodes.

Returns:

The imported node that belongs to this `Document`.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if the type of node being imported is not supported.

INVALID_CHARACTER_ERR: Raised if one of the imported names is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute. This may happen when importing an XML 1.1 [XML 1.1] element into an XML 1.0 document, for instance.

Since:

DOM Level 2

only Element & Attr has overloaded method with Namespace and Qualified name for node creation

---

createElementNS

```
Element createElementNS(java.lang.String namespaceURI,
                        java.lang.String qualifiedName)
                        throws DOMException
```

Creates an element of the given qualified name and namespace URI.
Per [XML Namespaces] , applications must use the value `null` as the namespaceURI parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the element to create.
`qualifiedName` - The qualified name of the element type to instantiate.

Returns:

A new `Element` object with the following attributes:

Attribute	Value
`Node.nodeName`	`qualifiedName`
`Node.namespaceURI`	`namespaceURI`
`Node.prefix`	prefix, extracted from `qualifiedName`, or `null` if there is no prefix
`Node.localName`	local name, extracted from `qualifiedName`
`Element.tagName`	`qualifiedName`

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified `qualifiedName` is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.
NAMESPACE_ERR: Raised if the `qualifiedName` is a malformed qualified name, if the `qualifiedName` has a prefix and the `namespaceURI` is `null`, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace" [XML Namespaces] , or if the `qualifiedName` or its prefix is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/", or if the `namespaceURI` is "http://www.w3.org/2000/xmlns/" and neither the `qualifiedName` nor its prefix is "xmlns".

NOT_SUPPORTED_ERR: Always thrown if the current document does not support the `"XML"` feature, since namespaces were defined by XML.

Since:

DOM Level 2

---

createAttributeNS

only Element & Attr has overloaded method with Namespace and Qualified name for node creation

```
Attr createAttributeNS(java.lang.String namespaceURI,
                       java.lang.String qualifiedName)
                       throws DOMException
```

Creates an attribute of the given qualified name and namespace URI.
Per [XML Namespaces] , applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the attribute to create.

`qualifiedName` - The qualified name of the attribute to instantiate.

Returns:

A new `Attr` object with the following attributes:

Attribute	Value
`Node.nodeName`	qualifiedName
`Node.namespaceURI`	`namespaceURI`
`Node.prefix`	prefix, extracted from `qualifiedName`, or `null` if there is no prefix
`Node.localName`	local name, extracted from `qualifiedName`
`Attr.name`	`qualifiedName`
`Node.nodeValue`	the empty string

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified `qualifiedName` is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.
NAMESPACE_ERR: Raised if the `qualifiedName` is a malformed qualified name, if the `qualifiedName` has a prefix and the `namespaceURI` is `null`, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from " http://www.w3.org/XML/1998/namespace", if the `qualifiedName` or its prefix is "xmls" and the

`namespaceURI` is different from "http://www.w3.org/2000/xmlns/", or if the `namespaceURI` is "http://www.w3.org/2000/xmlns/" and neither the `qualifiedName` nor its prefix is "xmlns".
NOT_SUPPORTED_ERR: Always thrown if the current document does not support the `"XML"` feature, since namespaces were defined by XML.

Since:

DOM Level 2

Search method will take Local name

Creation method take qualified name

---

getElementsByTagNameNS

this is the only has local name

```
NodeList getElementsByTagNameNS(java.lang.String namespaceURI,
                                java.lang.String localName)
```

Returns a `NodeList` of all the `Elements` with a given local name and namespace URI in document order.

Parameters:

`namespaceURI` - The namespace URI of the elements to match on. The special value `"*"` matches all namespaces.
`localName` - The local name of the elements to match on. The special value "*" matches all local names.

Returns:

A new `NodeList` object containing all the matched `Elements`.

Since:

DOM Level 2

---

getElementById

```
Element getElementById(java.lang.String elementId)
```

Returns the `Element` that has an ID attribute with the given value. If no such element exists, this returns `null` . If more than one element has an ID attribute with that value, what is returned is undefined.
The DOM implementation is expected to use the attribute `Attr.isId` to determine if an attribute is of type ID.

Note: Attributes with the name "ID" or "id" are not of type ID unless so defined.

Parameters:

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

Returns:

The matching element or `null` if there is none.

Since:

DOM Level 2

---

getInputEncoding

`java.lang.String` **`getInputEncoding`**`()`

it will return non-null regardless of XML declaration

An attribute specifying the encoding used for this document at the time of the parsing. This is `null` when it is not known, such as when the `Document` was created in memory.

Since:

DOM Level 3

---

getXmlEncoding

`java.lang.String` **`getXmlEncoding`**`()`

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

this will return non-null if and only if declared in XML file

Since:

DOM Level 3

~~getXmlStandalone~~

this is usefull only, if i use DTD. So, i DONT WANT

~~`boolean` **`getXmlStandalone`**`()`~~

~~An attribute specifying, as part of the XML declaration, whether this document is standalone. This is `false` when unspecified.~~

~~**Note:** No verification is done on the value when setting this attribute. Applications should use `Document.normalizeDocument()` with the "validate" parameter to verify if the value matches~~

the validity constraint for standalone document declaration as defined in [XML 1.0].

Since:

DOM Level 3

---

setXmlStandalone

this is usefull only, if i use DTD.
So, i DONT WANT

```
void setXmlStandalone(boolean xmlStandalone)
                      throws DOMException
```

An attribute specifying, as part of the XML declaration, whether this document is standalone. This is `false` when unspecified.

Note: No verification is done on the value when setting this attribute. Applications should use `Document.normalizeDocument()` with the "validate" parameter to verify if the value matches the validity constraint for standalone document declaration as defined in [XML 1.0].

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if this document does not support the "XML" feature.

Since:

DOM Level 3

---

getXmlVersion

```
java.lang.String getXmlVersion()
```

An attribute specifying, as part of the XML declaration, the version number of this document. If there is no declaration and if this document supports the "XML" feature, the value is `"1.0"`. If this document does not support the "XML" feature, the value is always `null`. Changing this attribute will affect methods that check for invalid characters in XML names. Application should invoke `Document.normalizeDocument()` in order to check for invalid characters in the `Node`s that are already part of this `Document`.
DOM applications may use the `DOMImplementation.hasFeature(feature, version)` method with parameter values "XMLVersion" and "1.0" (respectively) to determine if an implementation supports [XML 1.0]. DOM applications may use the same method with parameter values "XMLVersion" and "1.1" (respectively) to determine if an implementation supports [XML 1.1]. In both cases, in order to support XML, an implementation must also support the "XML" feature

defined in this specification. `Document` objects supporting a version of the "XMLVersion" feature must not raise a `NOT_SUPPORTED_ERR` exception for the same version number when using `Document.xmlVersion`.

Since:

DOM Level 3

---

setXmlVersion

```
void setXmlVersion(java.lang.String xmlVersion)
                   throws DOMException
```

An attribute specifying, as part of the XML declaration, the version number of this document. If there is no declaration and if this document supports the "XML" feature, the value is `"1.0"`. If this document does not support the "XML" feature, the value is always `null`. Changing this attribute will affect methods that check for invalid characters in XML names. Application should invoke `Document.normalizeDocument()` in order to check for invalid characters in the `Node`s that are already part of this `Document`.

DOM applications may use the `DOMImplementation.hasFeature(feature, version)` method with parameter values "XMLVersion" and "1.0" (respectively) to determine if an implementation supports [XML 1.0]. DOM applications may use the same method with parameter values "XMLVersion" and "1.1" (respectively) to determine if an implementation supports [XML 1.1]. In both cases, in order to support XML, an implementation must also support the "XML" feature defined in this specification. `Document` objects supporting a version of the "XMLVersion" feature must not raise a `NOT_SUPPORTED_ERR` exception for the same version number when using `Document.xmlVersion`.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if the version is set to a value that is not supported by this `Document` or if this document does not support the "XML" feature.

Since:

DOM Level 3

---

getStrictErrorChecking

```
boolean getStrictErrorChecking()
```

An attribute specifying whether error checking is enforced or not. When set to `false`, the implementation is free to not test every possible error case normally defined on DOM operations, and not raise any `DOMException` on DOM operations or report errors while using `Document.normalizeDocument()`. In case of error, the behavior is undefined. This attribute is `true` by default.

Since:

DOM Level 3

it does not mean WELL-FORMNESS error

---

setStrictErrorChecking

```
void setStrictErrorChecking(boolean strictErrorChecking)
```

An attribute specifying whether error checking is enforced or not. When set to `false`, the implementation is free to not test every possible error case normally defined on DOM operations, and not raise any `DOMException` on DOM operations or report errors while using `Document.normalizeDocument()`. In case of error, the behavior is undefined. This attribute is `true` by default.

Since:

DOM Level 3

---

getDocumentURI

so, we have to call setter method.

```
java.lang.String getDocumentURI()
```

The location of the document or `null` if undefined or if the `Document` was created using `DOMImplementation.createDocument`. No lexical checking is performed when setting this attribute; this could result in a `null` value returned when using `Node.baseURI`.
Beware that when the `Document` supports the feature "HTML" [DOM Level 2 HTML], the href attribute of the HTML BASE element takes precedence over this attribute when computing `Node.baseURI`.

Since:

DOM Level 3

---

setDocumentURI

if i give wrong location of document, it simply accept, wont throw any exception. i tested

```
void setDocumentURI(java.lang.String documentURI)
```

The location of the document or `null` if undefined or if the `Document` was created using `DOMImplementation.createDocument`. No lexical checking is performed when setting this attribute; this could result in a `null` value returned when using `Node.baseURI`.
Beware that when the `Document` supports the feature "HTML" [DOM Level 2 HTML], the href attribute of the HTML BASE element takes precedence over this attribute when computing `Node.baseURI`.

Since:

DOM Level 3

---

adoptNode

this method MOVE a node from one document to another document. where as, *importNode*() method COPY a node from one to another.

```
Node adoptNode(Node source)
               throws DOMException
```

Attempts to adopt a node from another document to this document. If supported, it changes the `ownerDocument` of the source node, its children, as well as the attached attribute nodes if there are any. If the source node has a parent it is first removed from the child list of its parent. This effectively allows moving a subtree from one document to another (unlike `importNode()` which create a copy of the source node instead of moving it). When it fails, applications should use `Document.importNode()` instead. Note that if the adopted node is already part of this document (i.e. the source and target document are the same), this method still has the effect of removing the source node from the child list of its parent, if any. The following list describes the specifics for each type of node.

ATTRIBUTE_NODE
: The `ownerElement` attribute is set to `null` and the `specified` flag is set to `true` on the adopted `Attr`. The descendants of the source `Attr` are recursively adopted.

DOCUMENT_FRAGMENT_NODE
: The descendants of the source node are recursively adopted.

DOCUMENT_NODE
: `Document` nodes cannot be adopted.

DOCUMENT_TYPE_NODE
: `DocumentType` nodes cannot be adopted.

ELEMENT_NODE
: *Specified* attribute nodes of the source element are adopted. Default attributes are discarded, though if the document being adopted into defines default attributes for this element name, those are assigned. The descendants of the source element are recursively adopted.

ENTITY_NODE
: `Entity` nodes cannot be adopted.

ENTITY_REFERENCE_NODE
: Only the `EntityReference` node itself is adopted, the descendants are discarded, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.

NOTATION_NODE
: `Notation` nodes cannot be adopted.

PROCESSING_INSTRUCTION_NODE, TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE
: These nodes can all be adopted. No specifics.

Note: Since it does not create new nodes unlike the `Document.importNode()` method, this method does not raise an `INVALID_CHARACTER_ERR` exception, and applications should use the `Document.normalizeDocument()` method to check if an imported name is not an XML name according to the XML version in use.

Parameters:
: `source` - The node to move into this document.

Returns:
: The adopted node, or `null` if this operation fails, such as when the source node comes from a different implementation.

Throws:
: `DOMException` - NOT_SUPPORTED_ERR: Raised if the source node is of type `DOCUMENT`, `DOCUMENT_TYPE`.
NO_MODIFICATION_ALLOWED_ERR: Raised when the source node is readonly.

Since:
: DOM Level 3

---

getDomConfig

`DOMConfiguration` **`getDomConfig()`**

The configuration used when `Document.normalizeDocument()` is invoked.

Since:
: DOM Level 3

So, Normalization is only for TWO (Text, EntityReference) node

normalizeDocument

```
void normalizeDocument()
```

This method acts as if the document was going through a save and load cycle, putting the document in a "normal" form. As a consequence, this method updates the replacement tree of `EntityReference` nodes and normalizes `Text` nodes, as defined in the method `Node.normalize()`.
Otherwise, the actual result depends on the features being set on the `Document.domConfig` object and governing what operations actually take place. Noticeably this method could also make the document namespace well-formed according to the algorithm described in , check the character normalization, remove the `CDATASection` nodes, etc. See `DOMConfiguration` for details.

FURTHER, we can configure

1. remove CDATASection
2. check nameSpace
3. Character normalization

```
// Keep in the document
 the information defined // in the XML Information
example)
 DOMConfiguration docConfig = myDocument.getDomCo
 docConfig.setParameter("infoset", Boolean.TRUE);
 myDocument.normalizeDocument();
```

Mutation events, when supported, are generated to reflect the changes occurring on the document.
If errors occur during the invocation of this method, such as an attempt to update a read-only node or a `Node.nodeName` contains an invalid character according to the XML version in use, errors or warnings (`DOMError.SEVERITY_ERROR` or `DOMError.SEVERITY_WARNING`) will be reported using the `DOMErrorHandler` object associated with the "error-handler " parameter. Note this method might also report fatal errors (`DOMError.SEVERITY_FATAL_ERROR`) if an implementation cannot recover from an error.

Since:

DOM Level 3

---

renameNode

```
Node renameNode(Node n,
                java.lang.String namespaceURI,
                java.lang.String qualifiedName)
                throws DOMException
```

only these 2. (other node does not have name for node right :)

Rename an existing node of type `ELEMENT_NODE` or `ATTRIBUTE_NODE`.

When possible this simply changes the name of the given node, otherwise this creates a new node with the specified name and replaces the existing node with the new node as described below.
If simply changing the name of the given node is not possible, the following operations are performed: a new node is created, any registered event listener is registered on the new node, any user data attached to the old node is removed from that node, the old node is removed from its parent if it has one, the children are moved to the new node, if the renamed node is an `Element` its attributes are moved to the new node, the new node is inserted at the position the old node used to have in its parent's child nodes list if it has one, the user data that was attached to the old node is attached to the new node.
When the node being renamed is an `Element` only the specified attributes are moved, default attributes originated from the DTD are updated according to the new element name. In addition, the implementation may update default attributes from other schemas. Applications should use `Document.normalizeDocument()` to guarantee these attributes are up-to-date.
When the node being renamed is an `Attr` that is attached to an `Element`, the node is first removed from the `Element` attributes map. Then, once renamed, either by modifying the existing node or creating a new one as described above, it is put back.
In addition,

- a user data event `NODE_RENAMED` is fired,
- when the implementation supports the feature "MutationNameEvents", each mutation operation involved in this method fires the appropriate event, and in the end the event { `http://www.w3.org/2001/xml-events`, `DOMElementNameChanged`} or { `http://www.w3.org/2001/xml-events`, `DOMAttributeNameChanged`} is fired.

Parameters:

`n` - The node to rename.
`namespaceURI` - The new namespace URI.
`qualifiedName` - The new qualified name.

Returns:

The renamed node. This is either the specified node or the new node that was created to replace the specified node.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised when the type of the specified node is neither `ELEMENT_NODE` nor `ATTRIBUTE_NODE`, or if the implementation does not support the renaming of the document element.
INVALID_CHARACTER_ERR: Raised if the new qualified name is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.
WRONG_DOCUMENT_ERR: Raised when the specified node was created from a different document than this document.
NAMESPACE_ERR: Raised if the `qualifiedName` is a malformed qualified name, if the `qualifiedName` has a prefix and the `namespaceURI` is `null`, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace" [XML Namespaces] . Also raised, when the node

being renamed is an attribute, if the `qualifiedName`, or its prefix, is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/".

Since:

DOM Level 3

---

13 - Dec - 08

it has 13 methods

5 - are overloaded parse methods
3 - methods for getter feature which set in Factory
1 - get Implementation
1 - create new Document
1 - get Schema object
1 - set SAX's ErrorHandler
1 - reset method

javax.xml.parsers

Class DocumentBuilder

```
java.lang.Object
  └ javax.xml.parsers.DocumentBuilder
```

```
public abstract class DocumentBuilder
```

extends java.lang.Object

Defines the API to obtain DOM Document instances from an XML document. Using this class, an application programmer can obtain a `Document` from XML.

An instance of this class can be obtained from the `DocumentBuilderFactory.newDocumentBuilder()` method. Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

Note that this class reuses several classes from the SAX API. ~~This does not require that the implementor of the underlying DOM implementation use a SAX parser to parse XML document into a `Document`. It merely requires that the implementation communicate with the application using these existing APIs.~~

Version:

$Revision: 1.6 $, $Date: 2006/07/12 14:18:07 $

Author:

Jeff Suttor

Constructor Summary

protected	**DocumentBuilder**() Protected constructor

Method Summary

abstract DOMImplementation	**getDOMImplementation**() Obtain an instance of a DOMImplementation object.
Schema	**getSchema**() Get a reference to the the Schema being used by the XML processor.
abstract boolean	**isNamespaceAware**() Indicates whether or not this gured to understand namespaces.
abstract boolean	**isValidating**() Indicates whether or not this gured to validate XML documents.
boolean	**isXIncludeAware**() Get the XInclude processing mode for this parser.
abstract Document	**newDocument**() Obtain a new instance of a DOM Document object to build a DOM tree with.
Document	**parse**(java.io.File f) Parse the content of the given file as an XML document and return a new DOM Document object.
abstract Document	**parse**(InputSource is) Parse the content of the given input source as an XML document and return a new DOM Document object.
Document	**parse**(java.io.InputStream is) Parse the content of the given InputStream as an XML document and return a new DOM Document object.
Document	**parse**(java.io.InputStream is, java.lang.String systemId) Parse the content of the given InputStream as an XML document and return a new DOM Document object.

these there three (is) methods available in SAXParse also for the same purpose.

use this method as much as possible. it is best practice

Document	**parse**(java.lang.String uri) Parse the content of the given URI as an XML document and return a new DOM Document object.
void	**reset**() Reset this DocumentBuilder to its original configuration.
abstract void	**setEntityResolver**(EntityResolver er) Specify the EntityResolver to be used to resolve entities present in the XML document to be parsed.
abstract void	**setErrorHandler**(ErrorHandler eh) Specify the ErrorHandler to be used by the pa

no getter method for this ErrorHandler

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DocumentBuilder

```
protected DocumentBuilder()
```

Protected constructor

Method Detail

reset

```
public void reset()
```

Reset this DocumentBuilder to its original configuration.

DocumentBuilder is reset to the same state as when it was created with DocumentBuilderFactory.newDocumentBuilder(). reset() is designed to allow

the reuse of existing `DocumentBuilder`s thus saving resources associated with the creation of new `DocumentBuilder`s.

The reset `DocumentBuilder` is not guaranteed to have the same `EntityResolver` or `ErrorHandler` `Object`s, e.g. `Object.equals(Object obj)`. It is guaranteed to have a functionally equal `EntityResolver` and `ErrorHandler`.

Throws:

`java.lang.UnsupportedOperationException` - When implementation does not override this method.

Since:

1.5

---

parse

```
public Document parse(java.io.InputStream is)
               throws SAXException,
                      java.io.IOException
```

Parse the content of the given `InputStream` as an XML document and return a new DOM `Document` object. An `IllegalArgumentException` is thrown if the `InputStream` is null.

Parameters:

`is` - InputStream containing the content to be parsed.

Returns:

`Document` result of parsing the `InputStream`

Throws:

`java.io.IOException` - If any IO errors occur.

`SAXException` - If any parse errors occur.

`java.lang.IllegalArgumentException` - When `is` is `null`

See Also:

`DocumentHandler`

---

parse

```
public Document parse(java.io.InputStream is,
                      java.lang.String systemId)
               throws SAXException,
                      java.io.IOException
```

Parse the content of the given `InputStream` as an XML document and return a new DOM `Document` object. An `IllegalArgumentException` is thrown if the `InputStream` is null.

Parameters:

`is` - InputStream containing the content to be parsed.

`systemId` - Provide a base for resolving relative URIs.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.

`SAXException` - If any parse errors occur.

`java.lang.IllegalArgumentException` - When `is` is `null`

See Also:

`DocumentHandler`

---

parse

```
public Document parse(java.lang.String uri)
               throws SAXException,
                      java.io.IOException
```

Parse the content of the given URI as an XML document and return a new DOM `Document` object. An `IllegalArgumentException` is thrown if the URI is `null` null.

Parameters:

`uri` - The location of the content to be parsed.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.

`SAXException` - If any parse errors occur.

`java.lang.IllegalArgumentException` - When `uri` is `null`

See Also:

`DocumentHandler`

---

parse

```
public Document parse(java.io.File f)
               throws SAXException,
                      java.io.IOException
```

Parse the content of the given file as an XML document and return a new DOM `Document` object. An `IllegalArgumentException` is thrown if the `File` is `null` null.

Parameters:

`f` - The file containing the XML to parse.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.

`SAXException` - If any parse errors occur.

`java.lang.IllegalArgumentException` - When `f` is `null`

See Also:

`DocumentHandler`

---

parse

```
public abstract Document parse(InputSource is)
                        throws SAXException,
                               java.io.IOException
```

Parse the content of the given input source as an XML document and return a new DOM `Document` object. An `IllegalArgumentException` is thrown if the `InputSource` is `null` null.

Parameters:

`is` - InputSource containing the content to be parsed.

Returns:

A new DOM Document object.

Throws:

`java.io.IOException` - If any IO errors occur.

`SAXException` - If any parse errors occur.

`java.lang.IllegalArgumentException` - When `is` is `null`

See Also:

`DocumentHandler`

---

isNamespaceAware

```
public abstract boolean isNamespaceAware()
```

Indicates whether or not this parser is configured to understand namespaces.

Returns:

true if this parser is configured to understand namespaces; false otherwise.

---

isValidating

```
public abstract boolean isValidating()
```

Indicates whether or not this parser is configured to validate XML documents.

Returns:

true if this parser is configured to validate XML documents; false otherwise.

---

setEntityResolver

```
public abstract void setEntityResolver(EntityResolver er)
```

Specify the `EntityResolver` to be used to resolve entities present in the XML document to be parsed. Setting this to `null` will result in the underlying implementation using it's own

default implementation and behavior.

Parameters:

`er` - The `EntityResolver` to be used to resolve entities present in the XML document to be parsed.

---

setErrorHandler

```
public abstract void setErrorHandler(ErrorHandler eh)
```

Specify the `ErrorHandler` to be used by the parser. Setting this to `null` will result in the underlying implementation using it's own default implementation and behavior.

Parameters:

`eh` - The `ErrorHandler` to be used by the parser.

---

newDocument

```
public abstract Document newDocument()
```

Obtain a new instance of a DOM `Document` object to build a DOM tree with.

Returns:

A new instance of a DOM Document object.

---

getDOMImplementation

```
public abstract DOMImplementation getDOMImplementation()
```

Obtain an instance of a `DOMImplementation` object.

Returns:

A new instance of a `DOMImplementation`.

---

getSchema

`public Schema getSchema()`

Get a reference to the the `Schema` being used by the XML processor.

If no schema is being used, `null` is returned.

Returns:
`Schema` being used or `null` if none in use

Throws:
`java.lang.UnsupportedOperationException` - When implementation does not override this method

Since:
1.5

---

isXIncludeAware

`public boolean isXIncludeAware()`

Get the XInclude processing mode for this parser.

Returns:
the return value of the `DocumentBuilderFactory.isXIncludeAware()` when this parser was created from factory.

Throws:
`java.lang.UnsupportedOperationException` - When implementation does not override this method

Since:
1.5

See Also:
`DocumentBuilderFactory.setXIncludeAware(boolean)`

Here after, just evaluate only IMPORTANT methods not just like setter / getter methods (18 - June - 09)

13 - Dec - 08

Schema object can be set only in DocumentBuilderFactory for DOM. Nowhere else !!!

it has 21 methods

2 - getting instance
2 - getter / setter for Schema
2 - getter / setter for attribute
2 - getter / setter for Feature
2 - getter / setter for coallasing
2 - getter / setter for parse
2 - getter / setter for validation
2 - getter / setter for namespace aware
2 - getter / setter for XInclude
2 - getter / setter for ingnorable white space
1 - create new DocumentBuilder

javax.xml.parsers

Class DocumentBuilderFactory

LS API cannot use for Validation

```
java.lang.Object
  └ javax.xml.parsers.DocumentBuilderFactory
```

```
public abstract class DocumentBuilderFactory
```

extends java.lang.Object

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

Version:

$Revision: 1.6 $, $Date: 2006/05/19 01:08:43 $

Author:

Jeff Suttor, Neeraj Bajaj

it has
6 getter + 6 setter methods are boolean methods for enable / query the feature

this is one of best example, where we have to use Abstract class and Interface to achieve OOPS abstraction.

if we need any origin use Abstract class with static methods or Constructor.
if we need 100 % abstraction use java interface

Constructor Su

protected	Docume Pr

Method Summary

attribute and property are rep. the same

abstract java.lang.Object	**getAttribute**(java.lang.String name) Allows the user to retrieve specific attributes on the underlying implementation.
abstract boolean	**getFeature**(java.lang.String name) Get the state of the named feature.
Schema	**getSchema**() Gets the Schema object specified through the setSchema(Schema schema) method.

we cannot set any arbitrary file into Schema and associate that with factory. if the associated file is not correct it will give error (i tested)

boolean	**isCoalescing**() Indicates whether or not the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node.
boolean	**isExpandEntityReferences**() Indicates whether or not the factory is configured to produce parsers which expand entity reference nodes.
boolean	**isIgnoringComments**() Indicates whether or not the factory is configured to produce parsers which ignores comments.
boolean	**isIgnoringElementContentWhitespace**() Indicates whether or not the factory is configured to produce parsers which ignore ignorable whitespace in element content.
an	**isNamespaceAware**() Indicates whether or not the factory is configured to produce parsers which are namespace aware.
an	**isValidating**() Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.
an	**isXIncludeAware**() Get state of XInclude processing.
abstract DocumentBuilder	**newDocumentBuilder**() Creates a new instance of a DocumentBuilder using the currently configured parameters.
static DocumentBuilderFactory	**newInstance**() Obtain a new instance of a DocumentBuilderFactory.
static DocumentBuilderFactory	**newInstance**(java.lang.String factoryClassName, java.lang.ClassLoader classLoader) Obtain a new instance of a DocumentBuilderFactory from class name.
abstract void	**setAttribute**(java.lang.String name, java.lang.Object value) Allows the user to set specific attributes on the underlying implementation.
void	**setCoalescing**(boolean coalescing) Specifies that the parser produced by this code will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node.
void	**setExpandEntityReferences**(boolean expandEntityRef) Specifies that the parser produced by this code will expand entity reference nodes.

All Factory method are start with "new" and appended by class name

newSAXParesr()
newValidator()
newValidatorHandler()
newDocumentBuilder()
newTransformer() and etc....

abstract void	**setFeature**(java.lang.String name, boolean value) Set a feature for this DocumentBuilderFactory and DocumentBuilders created by this factory.
void	**setIgnoringComments**(boolean ignoreComments) Specifies that the parser produced by this code will ignore comments.
void	**setIgnoringElementContentWhitespace**(boolean whitespace) Specifies that the parsers created by this factory must eliminate whitespace in element content (sometimes known loosely as 'ignorable whitespace') when parsing XML documents (see XML Rec 2.10).
void	**setNamespaceAware**(boolean awareness) Specifies that the parser produced by this code will provide support for XML namespaces.
void	**setSchema**(Schema schema) Set the Schema to be used by parsers created from this factory.
void	**setValidating**(boolean validating) Specifies that the parser produced by this code will validate documents as they are parsed.
void	**setXIncludeAware**(boolean state) Set state of XInclude processing.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DocumentBuilderFactory

```
protected DocumentBuilderFactory()
```

Protected constructor to prevent instantiation. Use newInstance().

Method Detail

newInstance

```
public static DocumentBuilderFactory newInstance()
```

Obtain a new instance of a `DocumentBuilderFactory`. This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the `DocumentBuilderFactory` implementation class to load:

- Use the `javax.xml.parsers.DocumentBuilderFactory` system property.
- Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the system property defined above. The jaxp.properties file is read only once by the JAXP implementation and it's values are then cached for future use. If the file does not exist when the first attempt is made to read from it, no further attempts are made to check for its existence. It is not possible to change the value of any property in jaxp.properties after it has been read for the first time.
- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/javax.xml.parsers.DocumentBuilderFactory` in jars available to the runtime.
- Platform default `DocumentBuilderFactory` instance.

Once an application has obtained a reference to a `DocumentBuilderFactory` it can use the factory to configure and obtain parser instances.

Tip for Trouble-shooting

Setting the `jaxp.debug` system property will cause this method to print a lot of debug messages to `System.err` about what it is doing and where it is looking at.

If you have problems loading `DocumentBuilder`s, try:

```
java -Djaxp.debug=1 YourProgram ....
```

Returns:

New instance of a `DocumentBuilderFactory`

Throws:

`FactoryConfigurationError` - if the implementation is not available or cannot be instantiated.

---

newInstance

```
public static DocumentBuilderFactory newInstance(java.lang.String factoryClassName,
                                                 java.lang.ClassLoader classLoader)
```

Obtain a new instance of a `DocumentBuilderFactory` from class name. This function is useful when there are multiple providers in the classpath. It gives more control to the application as it can specify which provider should be loaded.

Once an application has obtained a reference to a `DocumentBuilderFactory` it can use the factory to configure and obtain parser instances.

Tip for Trouble-shooting

Setting the `jaxp.debug` system property will cause this method to print a lot of debug messages to `System.err` about what it is doing and where it is looking at.

If you have problems try:

```
java -Djaxp.debug=1 YourProgram ....
```

Parameters:

`factoryClassName` - fully qualified factory class name that provides implementation of `javax.xml.parsers.DocumentBuilderFactory`.

`classLoader` - `ClassLoader` used to load the factory class. If `null` current `Thread`'s context classLoader is used to load the factory class.

Returns:

New instance of a `DocumentBuilderFactory`

Throws:

`FactoryConfigurationError` - if `factoryClassName` is `null`, or the factory class cannot be loaded, instantiated.

Since:

1.6

See Also:

`newInstance()`

---

newDocumentBuilder

```
public abstract DocumentBuilder newDocumentBuilder()
                                              throws ParserConfigurationException
```

Creates a new instance of a `DocumentBuilder` using the currently configured parameters.

Returns:

A new instance of a DocumentBuilder.

Throws:

`ParserConfigurationException` - if a DocumentBuilder cannot be created which satisfies the configuration requested.

---

setNamespaceAware

```
public void setNamespaceAware(boolean awareness)
```

Specifies that the parser produced by this code will provide support for XML namespaces. By default the value of

this is set to `false`

Parameters:

`awareness` - true if the parser produced will provide support for XML namespaces; false otherwise.

---

setValidating

```
public void setValidating(boolean validating)
```

Specifies that the parser produced by this code will validate documents as they are parsed. By default the value of this is set to `false`.

Note that "the validation" here means a validating parser as defined in the XML recommendation. In other words, it essentially just controls the DTD validation. (except the legacy two properties defined in JAXP 1.2.)

To use modern schema languages such as W3C XML Schema or RELAX NG instead of DTD, you can configure your parser to be a non-validating parser by leaving the `setValidating(boolean)` method `false`, then use the `setSchema(Schema)` method to associate a schema to a parser.

Parameters:

`validating` - true if the parser produced will validate documents as they are parsed; false otherwise.

---

setIgnoringElementContentWhitespace

```
public void setIgnoringElementContentWhitespace(boolean whitespace)
```

Specifies that the parsers created by this factory must eliminate whitespace in element content (sometimes known loosely as 'ignorable whitespace') when parsing XML documents (see XML Rec 2.10). Note that only whitespace which is directly contained within element content that has an element only content model (see XML Rec 3.2.1) will be eliminated. Due to reliance on the content model this setting requires the parser to be in validating mode. By default the value of this is set to `false`.

Parameters:

`whitespace` - true if the parser created must eliminate whitespace in the element content when parsing XML documents; false otherwise.

---

setExpandEntityReferences

```
public void setExpandEntityReferences(boolean expandEntityRef)
```

Specifies that the parser produced by this code will expand entity reference nodes. By default the value of this is set

to `true`

Parameters:

`expandEntityRef` - true if the parser produced will expand entity reference nodes; false otherwise.

---

setIgnoringComments

```
public void setIgnoringComments(boolean ignoreComments)
```

Specifies that the parser produced by this code will ignore comments. By default the value of this is set to `false` .

Parameters:

`ignoreComments` - `boolean` value to ignore comments during processing

---

setCoalescing

```
public void setCoalescing(boolean coalescing)
```

Specifies that the parser produced by this code will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node. By default the value of this is set to `false`

Parameters:

`coalescing` - true if the parser produced will convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node; false otherwise.

---

isNamespaceAware

```
public boolean isNamespaceAware()
```

Indicates whether or not the factory is configured to produce parsers which are namespace aware.

Returns:

true if the factory is configured to produce parsers which are namespace aware; false otherwise.

---

isValidating

```
public boolean isValidating()
```

Indicates whether or not the factory is configured to produce parsers which validate the XML content during parse.

Returns:

true if the factory is configured to produce parsers which validate the XML content during parse; false otherwise.

---

isIgnoringElementContentWhitespace

```
public boolean isIgnoringElementContentWhitespace()
```

Indicates whether or not the factory is configured to produce parsers which ignore ignorable whitespace in element content.

Returns:

true if the factory is configured to produce parsers which ignore ignorable whitespace in element content; false otherwise.

---

isExpandEntityReferences

```
public boolean isExpandEntityReferences()
```

Indicates whether or not the factory is configured to produce parsers which expand entity reference nodes.

Returns:

true if the factory is configured to produce parsers which expand entity reference nodes; false otherwise.

---

isIgnoringComments

```
public boolean isIgnoringComments()
```

Indicates whether or not the factory is configured to produce parsers which ignores comments.

Returns:

true if the factory is configured to produce parsers which ignores comments; false otherwise.

---

isCoalescing

```
public boolean isCoalescing()
```

Indicates whether or not the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node.

Returns:

true if the factory is configured to produce parsers which converts CDATA nodes to Text nodes and appends it to the adjacent (if any) Text node; false otherwise.

---

setAttribute

```
public abstract void setAttribute(java.lang.String name,
                                  java.lang.Object value)
                           throws java.lang.IllegalArgumentException
```

Allows the user to set specific attributes on the underlying implementation.

Parameters:

`name` - The name of the attribute.

`value` - The value of the attribute.

Throws:

`java.lang.IllegalArgumentException` - thrown if the underlying implementation doesn't recognize the attribute.

---

getAttribute

```
public abstract java.lang.Object getAttribute(java.lang.String name)
                                       throws java.lang.IllegalArgumentException
```

Allows the user to retrieve specific attributes on the underlying implementation.

Parameters:

`name` - The name of the attribute.

Returns:

value The value of the attribute.

Throws:

`java.lang.IllegalArgumentException` - thrown if the underlying implementation doesn't recognize the attribute.

---

setFeature

```
public abstract void setFeature(java.lang.String name,
                                boolean value)
                         throws ParserConfigurationException
```

Set a feature for this `DocumentBuilderFactory` and `DocumentBuilder`s created by this factory.

Feature names are fully qualified `URI`s. Implementations may define their own features. A `ParserConfigurationException` is thrown if this `DocumentBuilderFactory` or the `DocumentBuilder`s it creates cannot support the feature. It is possible for a `DocumentBuilderFactory` to expose a feature value but be unable to change its state.

All implementations are required to support the `XMLConstants.FEATURE_SECURE_PROCESSING` feature. When the feature is:

- `true`: the implementation will limit XML processing to conform to implementation limits. Examples include enity expansion limits and XML Schema constructs that would consume large amounts of resources. If XML processing is limited for security reasons, it will be reported via a call to the registered `ErrorHandler.fatalError(SAXParseException exception)`. See `DocumentBuilder.setErrorHandler(org.xml.sax.ErrorHandler errorHandler)`.
- `false`: the implementation will processing XML according to the XML specifications without regard to possible implementation limits.

Parameters:

`name` - Feature name.

`value` - Is feature state `true` or `false`.

Throws:

`ParserConfigurationException` - if this `DocumentBuilderFactory` or the `DocumentBuilder`s it creates cannot support this feature.

`java.lang.NullPointerException` - If the `name` parameter is null.

---

getFeature

```
public abstract boolean getFeature(java.lang.String name)
                            throws ParserConfigurationException
```

Get the state of the named feature.

Feature names are fully qualified `URI`s. Implementations may define their own features. An `ParserConfigurationException` is thrown if this `DocumentBuilderFactory` or the `DocumentBuilder`s it creates cannot support the feature. It is possible for an `DocumentBuilderFactory` to expose a feature value but be unable to change its state.

Parameters:

`name` - Feature name.

Returns:

State of the named feature.

Throws:

`ParserConfigurationException` - if this `DocumentBuilderFactory` or the `DocumentBuilder`s it creates cannot support this feature.

---

getSchema

`public Schema getSchema()`

Gets the `Schema` object specified through the `setSchema(Schema schema)` method.

Returns:
the `Schema` object that was last set through the `setSchema(Schema)` method, or null if the method was not invoked since a `DocumentBuilderFactory` is created.

Throws:
`java.lang.UnsupportedOperationException` - When implementation does not override this method.

Since:
1.5

---

setSchema

`public void setSchema(Schema schema)`

Set the `Schema` to be used by parsers created from this factory.

When a `Schema` is non-null, a parser will use a validator created from it to validate documents before it passes information down to the application.

When errors are found by the validator, the parser is responsible to report them to the user-specified `ErrorHandler` (or if the error handler is not set, ignore them or throw them), just like any other errors found by the parser itself. In other words, if the user-specified `ErrorHandler` is set, it must receive those errors, and if not, they must be treated according to the implementation specific default error handling rules.

A validator may modify the outcome of a parse (for example by adding default values that were missing in documents), and a parser is responsible to make sure that the application will receive modified DOM trees.

Initialy, null is set as the `Schema`.

This processing will take effect even if the `isValidating()` method returns `false`.

It is an error to use the `http://java.sun.com/xml/jaxp/properties/schemaSource` property and/or the `http://java.sun.com/xml/jaxp/properties/schemaLanguage` property in conjunction with a `Schema` object. Such configuration will cause a `ParserConfigurationException` exception when the `newDocumentBuilder()` is invoked.

Note for implmentors

A parser must be able to work with any `Schema` implementation. However, parsers and schemas are allowed to use implementation-specific custom mechanisms as long as they yield the result described in the specification.

Parameters:
`schema` - `Schema` to use or `null` to remove a schema.
Throws:
`java.lang.UnsupportedOperationException` - When implementation does not override this method.
Since:
1.5

---

setXIncludeAware

```
public void setXIncludeAware(boolean state)
```

Set state of XInclude processing.

If XInclude markup is found in the document instance, should it be processed as specified in XML Inclusions (XInclude) Version 1.0.

XInclude processing defaults to `false`.

Parameters:
`state` - Set XInclude processing to `true` or `false`
Throws:
`java.lang.UnsupportedOperationException` - When implementation does not override this method.
Since:
1.5

---

isXIncludeAware

```
public boolean isXIncludeAware()
```

Get state of XInclude processing.

Returns:
current state of XInclude processing
Throws:
`java.lang.UnsupportedOperationException` - When implementation does not override this method.

Since:

1.5

---

18 - Dec - 2008

No methods

org.w3c.dom

Interface DocumentFragment

All Superinterfaces:

Node

`public interface` **`DocumentFragment`**

extends Node

`DocumentFragment` is a "lightweight" or "minimal" `Document` object. It is very common to want to be able to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments and it is quite natural to use a Node for this purpose. While it is true that a `Document` object could fulfill this role, a `Document` object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. `DocumentFragment` is such an object.

Furthermore, various operations -- such as inserting nodes as children of another `Node` -- may take `DocumentFragment` objects as arguments; this results in all the child nodes of the `DocumentFragment` being moved to the child list of this node.

so many child lly

The children of a `DocumentFragment` node are zero or more nodes representing the tops of any sub-trees defining the structure of the document. `DocumentFragment` nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a `DocumentFragment` might have only one child and that child node could be a `Text` node. Such a structure model represents neither an HTML document nor a well-formed XML document.

very.... very... super. SLOW study

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`. This makes the `DocumentFragment` very useful when the user wishes to

create nodes that are siblings; the `DocumentFragment` acts as the parent of these nodes so that the user can use the standard methods from the `Node` interface, such as `Node.insertBefore` and `Node.appendChild`.

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

---

21 - Oct - 08

19 - Dec - 08

org.w3c.dom.traversal

Interface DocumentTraversal

only 2 methods

it act as almost Factory for creating TreeWalker / NodeIterator

Tree walker can be ONLY used with SubTree. NodeIterator can be subtree / nodelist /xpath query result

```
public interface DocumentTraversal
```

so, i can cast Document object to DocumentTraversal and start to use

`DocumentTraversal` contains methods that create `NodeIterators` and `TreeWalkers` to traverse a node and its children in document order (depth first, pre-order traversal, which is equivalent to the order in which the start tags occur in the text representation of the document). In DOMs which support the Traversal feature, `DocumentTraversal` will be implemented by the same objects that implement the Document interface.

See also the Document Object Model (DOM) Level 2 Traversal and Range Specification.

Since:

DOM Level 2

Method Summary

NodeIterator	**createNodeIterator**(Node root, int whatToShow, NodeFilter filter, boolean entityReferenceExpansion) Create a new NodeIterator over the subtree rooted at the specified node.
TreeWalker	**createTreeWalker**(Node root, int whatToShow, NodeFilter filter, boolean entityReferenceExpansion) Create a new TreeWalker over the subtree rooted at the specified node.

both method has same parameters

Method Detail

createNodeIterator

```
NodeIterator createNodeIterator(Node root,
                                 int whatToShow,
                                 NodeFilter filter,
                                 boolean entityReferenceExpansion)
                                 throws DOMException
```

At a time, it can get logical view of SINGLE type of node. Either, Element / Comment / CDATASection / Text

Create a new `NodeIterator` over the subtree rooted at the specified node.

Parameters:

`root` - The node which will be iterated together with its children. The `NodeIterator` is initially positioned just before this node. The `whatToShow` flags and the filter, if any, are not considered when setting this position. The root must not be `null`.

`whatToShow` - This flag specifies which node types may appear in the logical view of the tree presented by the `NodeIterator`. See the description of `NodeFilter` for the set of possible `SHOW_` values.These flags can be combined using `OR`.

`filter` - The `NodeFilter` to be used with this `NodeIterator`, or `null` to indicate no filter.

`entityReferenceExpansion` - The value of this flag determines whether entity reference nodes are expanded.

Returns:

The newly created `NodeIterator`.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if the specified `root` is `null`.

---

createTreeWalker

```
TreeWalker createTreeWalker(Node root,
                            int whatToShow,
                            NodeFilter filter,
                            boolean entityReferenceExpansion)
                            throws DOMException
```

Create a new `TreeWalker` over the subtree rooted at the specified node.

Parameters:

 what means currentNode ?

`root` - The node which will serve as the `root` for the `TreeWalker`. The `whatToShow` flags and the `NodeFilter` are not considered when setting this value; any node type will be accepted as the `root`. The `currentNode` of the `TreeWalker` is initialized to this node, whether or not it is visible. The `root` functions as a stopping point for traversal methods that look upward in the document structure, such as `parentNode` and nextNode. The `root` must not be `null`.

`whatToShow` - This flag specifies which node types may appear in the logical view of the tree presented by the `TreeWalker`. See the description of `NodeFilter` for the set of possible `SHOW_` values.These flags can be combined using `OR`.

`filter` - The `NodeFilter` to be used with this `TreeWalker`, or `null` to indicate no filter.

`entityReferenceExpansion` - If this flag is false, the contents of `EntityReference` nodes are not presented in the logical view.

Returns:

The newly created `TreeWalker`.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if the specified `root` is `null`.

At 5th round get familier with DOMConfiguration object.
it is used by Classes of DOM are
1. LSSerializer
2. LSParser
3. DOcument

17 - Oct - 08

SAX also, would have a object to hold its configuration parameters. but SAX dont have as DOM

it has only four methods

org.w3c.dom

Interface DOMConfiguration

Mostly DOMString and DOMStringList objects are used for represention of various DOM configuration parameters

`public interface DOMConfiguration`

The `DOMConfiguration` interface represents the configuration of a document and maintains a table of recognized parameters. Using the configuration, it is possible to change `Document.normalizeDocument()` behavior, such as replacing the `CDATASection` nodes with `Text` nodes or specifying the type of the schema that must be used when the validation of the `Document` is requested. `DOMConfiguration` objects are also used in [DOM Level 3 Load and Save] in the `DOMParser` and `DOMSerializer` interfaces.

The parameter names used by the `DOMConfiguration` object are defined throughout the DOM Level 3 specifications. Names are case-insensitive. To avoid possible conflicts, as a convention, names referring to parameters defined outside the DOM specification should be made unique. Because parameters are exposed as properties in the , names are recommended to follow the section 5.16 Identifiers of [Unicode] with the addition of the character '-' (HYPHEN-MINUS) but it is not enforced by the DOM implementation. DOM Level 3 Core Implementations are required to recognize all parameters defined in this specification. Some parameter values may also be required to be supported by the implementation. Refer to the definition of the parameter to know if a value must be supported or not.

Note: Parameters are similar to features and properties used in SAX2 [SAX].

The following list of parameters defined in the DOM:

`"canonical-form"`

`true`

[*optional*] Canonicalize the document according to the rules specified in [Canonical XML], such as removing the `DocumentType` node (if any) from the tree, or removing superfluous namespace declarations from each element. Note that this is limited to what can be represented in the DOM; in particular, there is no way to specify the order of the attributes in the DOM. In addition, Setting this parameter to `true` will also set the state of

the parameters listed below. Later changes to the state of one of those parameters will revert "canonical-form" back to `false`. Parameters set to `false`: "entities", "normalize-characters", "cdata-sections". Parameters set to `true`: "namespaces", "namespace-declarations", "well-formed", "element-content-whitespace". Other parameters are not changed unless explicitly specified in the description of the parameters.

`false`

[*required*] (*default*)Do not canonicalize the document.

`"cdata-sections"`

`true`

[*required*] (*default*)Keep `CDATASection` nodes in the document.

`false`

[*required*]Transform `CDATASection` nodes in the document into `Text` nodes. The new `Text` node is then combined with any adjacent `Text` node.

`"check-character-normalization"`

`true`

[*optional*] Check if the characters in the document are fully normalized, as defined in appendix B of [XML 1.1]. When a sequence of characters is encountered that fails normalization checking, an error with the `DOMError.type` equals to "check-character-normalization-failure" is issued.

`false`

[*required*] (*default*)Do not check if characters are normalized.

`"comments"`

`true`

[*required*] (*default*)Keep `Comment` nodes in the document.

`false`

[*required*]Discard `Comment` nodes in the document.

`"datatype-normalization"`

`true`

[*optional*] Expose schema normalized values in the tree, such as XML Schema normalized values in the case of XML Schema. Since this parameter requires to have schema information, the "validate" parameter will also be set to `true`. Having this parameter activated when "validate" is `false` has no effect and no schema-normalization will happen.

Note: Since the document contains the result of the XML 1.0 processing, this parameter does not apply to attribute value normalization as defined in section 3.3.3 of [XML 1.0] and is only meant for schema languages other than Document Type Definition (DTD).

`false`

[*required*] (*default*) Do not perform schema normalization on the tree.

`"element-content-whitespace"`

`true`

[*required*] (*default*)Keep all whitespaces in the document.

`false`

[*optional*] Discard all `Text` nodes that contain whitespaces in element content, as described in [element content whitespace]. The implementation is expected to use the attribute `Text.isElementContentWhitespace` to determine if a `Text` node should be discarded or not.

`"entities"`

`true`

[*required*] (*default*)Keep `EntityReference` nodes in the document.

`false`

[*required*] Remove all `EntityReference` nodes from the document, putting the entity expansions directly in their place. `Text` nodes are normalized, as defined in `Node.normalize`. Only unexpanded entity references are kept in the document.

Note: This parameter does not affect `Entity` nodes.

`"error-handler"`

[*required*] Contains a `DOMErrorHandler` object. If an error is encountered in the document, the implementation will call back the `DOMErrorHandler` registered using this parameter. The implementation may provide a default `DOMErrorHandler` object. When called, `DOMError.relatedData` will contain the closest node to where the error occurred. If the implementation is unable to determine the node where the error occurs, `DOMError.relatedData` will contain the `Document` node. Mutations to the document from within an error handler will result in implementation dependent behavior.

`"infoset"`

`true`

[*required*]Keep in the document the information defined in the XML Information Set [XML Information Set] .This forces the following parameters to `false`: " validate-if-schema", "entities", "datatype-normalization", "cdata-sections ".This forces the following parameters to `true`: " namespace-declarations", "well-formed", "element-content-whitespace", " comments", "namespaces".Other parameters are not changed unless explicitly specified in the description of the parameters. Note that querying this parameter with `getParameter` returns `true` only if the individual parameters specified above are appropriately set.

`false`

Setting `infoset` to `false` has no effect.

`"namespaces"`

`true`

[*required*] (*default*) Perform the namespace processing as defined in .

`false`

[*optional*] Do not perform the namespace processing.

`"namespace-declarations"`

This parameter has no effect if the parameter "namespaces" is set to `false`.

`true`

[*required*] (*default*) Include namespace declaration attributes, specified or defaulted from the schema, in the document. See also the sections "Declaring Namespaces" in [XML Namespaces] and [XML Namespaces 1.1] .

`false`

[*required*]Discard all namespace declaration attributes. The namespace prefixes (`Node.prefix`) are retained even if this parameter is set to `false`.

`"normalize-characters"`

`true`

[*optional*] Fully normalized the characters in the document as defined in appendix B of [XML 1.1].

`false`

[*required*] (*default*)Do not perform character normalization.

`"schema-location"`

[*optional*] Represent a `DOMString` object containing a list of URIs, separated by whitespaces (characters matching the nonterminal production S defined in section 2.3 [XML 1.0]), that represents the schemas against which validation should occur, i.e. the current schema. The types of schemas referenced in this list must match the type specified with `schema-type`, otherwise the behavior of an implementation is undefined. The schemas specified using this property take precedence to the schema information specified in the document itself. For namespace aware schema, if a schema specified using this property and a schema specified in the document instance (i.e. using the `schemaLocation` attribute) in a schema document (i.e. using schema `import` mechanisms) share the same `targetNamespace`, the schema specified by the user using this property will be used. If two schemas specified using this property share the same `targetNamespace` or have no namespace, the behavior is implementation dependent. If no location has been provided, this parameter is `null`.

Note: The `"schema-location"` parameter is ignored unless the "schema-type" parameter value is set. It is strongly recommended that `Document.documentURI` will be set so that an implementation can successfully resolve any external entities referenced.

`"schema-type"`

[*optional*] Represent a `DOMString` object containing an absolute URI and representing the type of the schema language used to validate a document against. Note that no lexical checking is done on the absolute URI. If this parameter is not set, a default value may be provided by the implementation, based on the schema languages supported and on the schema language used at load time. If no value is provided, this parameter is `null`.

Note: For XML Schema [XML Schema Part 1] , applications must use the value `"http://`

`www.w3.org/2001/XMLSchema"`. For XML DTD [XML 1.0], applications must use the value `"http://www.w3.org/TR/REC-xml"`. Other schema languages are outside the scope of the W3C and therefore should recommend an absolute URI in order to use this method.

`"split-cdata-sections"`

`true`

[*required*] (*default*)Split CDATA sections containing the CDATA section termination marker ']]>'. When a CDATA section is split a warning is issued with a `DOMError.type` equals to `"cdata-sections-splitted"` and `DOMError.relatedData` equals to the first `CDATASection` node in document order resulting from the split.

`false`

[*required*]Signal an error if a `CDATASection` contains an unrepresentable character.

`"validate"`

`true`

[*optional*] Require the validation against a schema (i.e. XML schema, DTD, any other type or representation of schema) of the document as it is being normalized as defined by [XML 1.0]. If validation errors are found, or no schema was found, the error handler is notified. Schema-normalized values will not be exposed according to the schema in used unless the parameter "datatype-normalization" is `true`. This parameter will reevaluate:

- Attribute nodes with `Attr.specified` equals to `false`, as specified in the description of the `Attr` interface;
- The value of the attribute `Text.isElementContentWhitespace` for all `Text` nodes;
- The value of the attribute `Attr.isId` for all `Attr` nodes;
- The attributes `Element.schemaTypeInfo` and `Attr.schemaTypeInfo`.

Note: "validate-if-schema" and "validate" are mutually exclusive, setting one of them to `true` will set the other one to `false`. Applications should also consider setting the parameter "well-formed" to `true`, which is the default for that option, when validating the document.

`false`

[*required*] (*default*) Do not accomplish schema processing, including the internal subset processing. Default attribute values information are kept. Note that validation might still happen if "validate-if-schema" is `true` .

`"validate-if-schema"`

`true`

[*optional*]Enable validation only if a declaration for the document element can be found in a schema (independently of where it is found, i.e. XML schema, DTD, or any other type or representation of schema). If validation is enabled, this parameter has the same behavior as the parameter "validate" set to `true`.

Note: "validate-if-schema" and "validate" are mutually exclusive, setting one of them to `true` will set the other one to `false`.

`false`

[*required*] (*default*) No schema processing should be performed if the document has a schema, including internal subset processing. Default attribute values information are kept. Note that validation must still happen if "validate " is `true`.

`"well-formed"`

`true`

[*required*] (*default*) Check if all nodes are XML well formed according to the XML version in use in `Document.xmlVersion`:

- check if the attribute `Node.nodeName` contains invalid characters according to its node type and generate a `DOMError` of type `"wf-invalid-character-in-node-name"`, with a `DOMError.SEVERITY_ERROR` severity, if necessary;
- check if the text content inside `Attr`, `Element`, `Comment`, `Text`, `CDATASection` nodes for invalid characters and generate a `DOMError` of type `"wf-invalid-character"`, with a `DOMError.SEVERITY_ERROR` severity, if necessary;
- check if the data inside `ProcessingInstruction` nodes for invalid characters and generate a `DOMError` of type `"wf-invalid-character"`, with a `DOMError.SEVERITY_ERROR` severity, if necessary;

`false`

[*optional*] Do not check for XML well-formedness.

The resolution of the system identifiers associated with entities is done using `Document.documentURI`. However, when the feature "LS" defined in [DOM Level 3 Load and Save] is supported by the DOM implementation, the parameter "resource-resolver" can also be used on `DOMConfiguration` objects attached to `Document` nodes. If this parameter is set, `Document.normalizeDocument()` will invoke the resource resolver instead of using `Document.documentURI`.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

boolean	**canSetParameter**(java.lang.String name, java.lang.Object value) Check if setting a parameter to a specific value is supported.
java.lang.Object	**getParameter**(java.lang.String name) Return the value of a parameter if known.
DOMStringList	**getParameterNames**() The list of the parameters supported by this DOMConfiguration object and for which at least one value can be set by the application.
void	**setParameter**(java.lang.String name, java.lang.Object value) Set the value of a parameter.

Method Detail

setParameter

```
void setParameter(java.lang.String name,
                  java.lang.Object value)
                  throws DOMException
```

Set the value of a parameter.

Parameters:

name - The name of the parameter to set.

value - The new value or null if the user wishes to unset the parameter. While the type of the value parameter is defined as DOMUserData, the object type must match the type defined by the definition of the parameter. For example, if the parameter is "error-handler", the value must be of type DOMErrorHandler.

Throws:

DOMException - NOT_FOUND_ERR: Raised when the parameter name is not recognized.
NOT_SUPPORTED_ERR: Raised when the parameter name is recognized but the requested value cannot be set.
TYPE_MISMATCH_ERR: Raised if the value type for this parameter name is incompatible with the expected value type.

getParameter

```
java.lang.Object getParameter(java.lang.String name)
                              throws DOMException
```

Return the value of a parameter if known.

Parameters:

`name` - The name of the parameter.

Returns:

The current object associated with the specified parameter or `null` if no object has been associated or if the parameter is not supported.

Throws:

`DOMException` - NOT_FOUND_ERR: Raised when the parameter name is not recognized.

---

canSetParameter

```
boolean canSetParameter(java.lang.String name,
                        java.lang.Object value)
```

Check if setting a parameter to a specific value is supported.

Parameters:

`name` - The name of the parameter to check.

`value` - An object. if `null`, the returned value is `true`.

Returns:

`true` if the parameter could be successfully set to the specified value, or `false` if the parameter is not recognized or the requested value is not supported. This does not change the current value of the parameter itself.

---

getParameterNames

```
DOMStringList getParameterNames()
```

The list of the parameters supported by this `DOMConfiguration` object and for which at least one value can be set by the application. Note that this list can also contain parameter names defined outside this specification.

---

org.w3c.dom

Interface DOMError

it has 6 methods

i dont know how to execute. atleast 5th round find use net to get sample

why, error, simply exception is enough right ????... i dont know

i dont know answer at 4th round also

```
public interface DOMError
```

`DOMError` is an interface that describes an error.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Field Summary

static short	**SEVERITY_ERROR** The severity of the error described by the `DOMError` is error.
static short	**SEVERITY_FATAL_ERROR** The severity of the error described by the `DOMError` is fatal error.
static short	**SEVERITY_WARNING** The severity of the error described by the `DOMError` is warning.

Method Summary

DOMLocator	**getLocation**() The location of the error.
java.lang.String	**getMessage**() An implementation specific string describing the error that occurred.

java.lang.Object	**getRelatedData**() The related DOMError.type dependent data if any.
java.lang.Object	**getRelatedException**() The related platform dependent exception if any.
short	**getSeverity**() The severity of the error, either SEVERITY_WARNING, SEVERITY_ERROR, or SEVERITY_FATAL_ERROR.
java.lang.String	**getType**() A DOMString indicating which related data is expected in relatedData.

Field Detail

SEVERITY_WARNING

```
static final short SEVERITY_WARNING
```

The severity of the error described by the DOMError is warning. A SEVERITY_WARNING will not cause the processing to stop, unless DOMErrorHandler.handleError() returns false.

See Also:

Constant Field Values

SEVERITY_ERROR

```
static final short SEVERITY_ERROR
```

The severity of the error described by the DOMError is error. A SEVERITY_ERROR may not cause the processing to stop if the error can be recovered, unless DOMErrorHandler.handleError() returns false.

See Also:

Constant Field Values

SEVERITY_FATAL_ERROR

```
static final short SEVERITY_FATAL_ERROR
```

The severity of the error described by the `DOMError` is fatal error. A `SEVERITY_FATAL_ERROR` will cause the normal processing to stop. The return value of `DOMErrorHandler.handleError()` is ignored unless the implementation chooses to continue, in which case the behavior becomes undefined.

See Also:

Constant Field Values

Method Detail

getSeverity

```
short getSeverity()
```

The severity of the error, either `SEVERITY_WARNING`, `SEVERITY_ERROR`, or `SEVERITY_FATAL_ERROR`.

---

getMessage

```
java.lang.String getMessage()
```

An implementation specific string describing the error that occurred.

---

getType

```
java.lang.String getType()
```

A `DOMString` indicating which related data is expected in `relatedData`. Users should refer

to the specification of the error in order to find its `DOMString` type and `relatedData` definitions if any.

Note: As an example, `Document.normalizeDocument()` does generate warnings when the "split-cdata-sections" parameter is in use. Therefore, the method generates a `SEVERITY_WARNING` with `type` `"cdata-sections-splitted"` and the first `CDATASection` node in document order resulting from the split is returned by the `relatedData` attribute.

---

getRelatedException

`java.lang.Object` **`getRelatedException`**`()`

The related platform dependent exception if any.

---

getRelatedData

`java.lang.Object` **`getRelatedData`**`()`

The related `DOMError.type` dependent data if any.

---

getLocation

`DOMLocator` **`getLocation`**`()`

The location of the error.

---

17 - Oct - 08

org.w3c.dom

Interface DOMErrorHandler

only one methods

i dont know how to execute. atleast 5th round find use net to get sample

DOM can ALSO use SAX's ErrorHandler via DocumentBuilder

```
public interface DOMErrorHandler
```

`DOMErrorHandler` is a callback interface that the DOM implementation can call when reporting errors that happens while processing XML data, or when doing some other processing (e.g. validating a document). A `DOMErrorHandler` object can be attached to a `Document` using the "error-handler" on the `DOMConfiguration` interface. If more than one error needs to be reported during an operation, the sequence and numbers of the errors passed to the error handler are implementation dependent.

The application that is using the DOM implementation is expected to implement this interface.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

boolean	**handleError**(DOMError error) This method is called on the error handler when an error occurs.

Method Detail

handleError

`boolean` **`handleError`**`(DOMError error)`

This method is called on the error handler when an error occurs.
If an exception is thrown from this method, it is considered to be equivalent of returning `true`.

Parameters:

`error` - The error object that describes the error. This object may be reused by the DOM implementation across multiple calls to the `handleError` method.

Returns:

If the `handleError` method returns `false`, the DOM implementation should stop the current processing when possible. If the method returns `true`, the processing may continue depending on `DOMError.severity`.

---

17 - Oct - 08

Please read all the attributes of this inteface

org.w3c.dom

Class DOMException

17 important attribute or error code

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.lang.RuntimeException
              └org.w3c.dom.DOMException
```

All Implemented Interfaces:

java.io.Serializable

```
public class DOMException
```

extends java.lang.RuntimeException

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). In general, DOM methods return specific error values in ordinary processing situations, such as out-of-bound errors when using `NodeList`.

Implementations should raise other exceptions under other circumstances. For example, implementations should raise an implementation-dependent exception if a `null` argument is passed when `null` was not expected.

~~Some languages and object systems do not support the concept of exceptions. For such systems, error conditions may be indicated using native error reporting mechanisms. For some bindings, for example, methods may return error codes similar to those listed in the corresponding method descriptions.~~

See also the Document Object Model (DOM) Level 3 Core Specification.

See Also:

Serialized Form

Field Summary

short	**code**
static short	**DOMSTRING_SIZE_ERR** If the specified range of text does not fit into a `DOMString`.
static short	**HIERARCHY_REQUEST_ERR** If any `Node` is inserted somewhere it doesn't belong.
static short	**INDEX_SIZE_ERR** If index or size is negative, or greater than the allowed value.
static short	**INUSE_ATTRIBUTE_ERR** If an attempt is made to add an attribute that is already in use elsewhere.
static short	**INVALID_ACCESS_ERR** If a parameter or an operation is not supported by the underlying object.
static short	**INVALID_CHARACTER_ERR** If an invalid or illegal character is specified, such as in an XML name.
static short	**INVALID_MODIFICATION_ERR** If an attempt is made to modify the type of the underlying object.
static short	**INVALID_STATE_ERR** ex: delete a node , the try to get child of that deleted node If an attempt is made to use an object that is not, or is no longer, usable.
static short	**NAMESPACE_ERR** If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.
static short	**NO_DATA_ALLOWED_ERR** If data is specified for a `Node` which does not support data.
static short	**NO_MODIFICATION_ALLOWED_ERR** If an attempt is made to modify an object where modifications are not allowed.
static short	**NOT_FOUND_ERR** If an attempt is made to reference a `Node` in a context where it does not exist.

static short	**NOT_SUPPORTED_ERR** If the implementation does not support the requested type of object or operation.
static short	**SYNTAX_ERR** If an invalid or illegal string is specified.
static short	**TYPE_MISMATCH_ERR** If the type of an object is incompatible with the expected type of the parameter associated to the object.
static short	**VALIDATION_ERR** If a call to a method such as `insertBefore` or `removeChild` would make the `Node` invalid with respect to "partial validity", this exception would be raised and the operation would not be done.
static short	**WRONG_DOCUMENT_ERR** If a `Node` is used in a different document than the one that created it (that doesn't support it).

Constructor Summary

DOMException(short code, java.lang.String message)

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

code

`public short code`

INDEX_SIZE_ERR

`public static final short INDEX_SIZE_ERR`

If index or size is negative, or greater than the allowed value.

See Also:

Constant Field Values

DOMSTRING_SIZE_ERR

`public static final short DOMSTRING_SIZE_ERR`

If the specified range of text does not fit into a `DOMString`.

See Also:

Constant Field Values

HIERARCHY_REQUEST_ERR

`public static final short HIERARCHY_REQUEST_ERR`

If any `Node` is inserted somewhere it doesn't belong.

See Also:

Constant Field Values

WRONG_DOCUMENT_ERR

```
public static final short WRONG_DOCUMENT_ERR
```

If a `Node` is used in a different document than the one that created it (that doesn't support it).

See Also:

Constant Field Values

INVALID_CHARACTER_ERR

```
public static final short INVALID_CHARACTER_ERR
```

If an invalid or illegal character is specified, such as in an XML name.

See Also:

Constant Field Values

NO_DATA_ALLOWED_ERR

```
public static final short NO_DATA_ALLOWED_ERR
```

If data is specified for a `Node` which does not support data.

See Also:

Constant Field Values

NO_MODIFICATION_ALLOWED_ERR

```
public static final short NO_MODIFICATION_ALLOWED_ERR
```

If an attempt is made to modify an object where modifications are not allowed.

See Also:

Constant Field Values

NOT_FOUND_ERR

```
public static final short NOT_FOUND_ERR
```

If an attempt is made to reference a `Node` in a context where it does not exist.

See Also:
Constant Field Values

NOT_SUPPORTED_ERR

```
public static final short NOT_SUPPORTED_ERR
```

If the implementation does not support the requested type of object or operation.

See Also:
Constant Field Values

INUSE_ATTRIBUTE_ERR

```
public static final short INUSE_ATTRIBUTE_ERR
```

If an attempt is made to add an attribute that is already in use elsewhere.

See Also:
Constant Field Values

INVALID_STATE_ERR

`public static final short` **`INVALID_STATE_ERR`**

If an attempt is made to use an object that is not, or is no longer, usable.

Since:
DOM Level 2
See Also:
Constant Field Values

SYNTAX_ERR

`public static final short` **`SYNTAX_ERR`**

If an invalid or illegal string is specified.

Since:
DOM Level 2
See Also:
Constant Field Values

INVALID_MODIFICATION_ERR

`public static final short` **`INVALID_MODIFICATION_ERR`**

If an attempt is made to modify the type of the underlying object.

Since:
DOM Level 2
See Also:
Constant Field Values

NAMESPACE_ERR

`public static final short` **`NAMESPACE_ERR`**

If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

Since:
DOM Level 2
See Also:
Constant Field Values

INVALID_ACCESS_ERR

```
public static final short INVALID_ACCESS_ERR
```

If a parameter or an operation is not supported by the underlying object.

Since:
DOM Level 2
See Also:
Constant Field Values

VALIDATION_ERR

```
public static final short VALIDATION_ERR
```

If a call to a method such as `insertBefore` or `removeChild` would make the `Node` invalid with respect to "partial validity", this exception would be raised and the operation would not be done. This code is used in [DOM Level 3 Validation] . Refer to this specification for further information.

Since:
DOM Level 3
See Also:
Constant Field Values

TYPE_MISMATCH_ERR

```
public static final short TYPE_MISMATCH_ERR
```

If the type of an object is incompatible with the expected type of the parameter associated to the object.

Since:

DOM Level 3

See Also:

Constant Field Values

Constructor Detail

DOMException

```
public DOMException(short code,
                    java.lang.String message)
```

org.w3c.dom

Interface DOMImplementation

it has 2 methods

2 - getter / query for Feature

public interface **DOMImplementation**

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

See also the Document Object Model (DOM) Level 3 Core Specification.

Method Summary

Document	**createDocument**(java.lang.String namespaceURI, java.lang.String qualifiedName, DocumentType doctype) Creates a DOM Document object of the specified type with its document element.
DocumentType	**createDocumentType**(java.lang.String qualifiedName, java.lang.String publicId, java.lang.String systemId) Creates an empty DocumentType node.
java.lang.Object	**getFeature**(java.lang.String feature, java.lang.String version) This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in DOM Features.
boolean	**hasFeature**(java.lang.String feature, java.lang.String version) Test if the DOM implementation implements a specific feature and version, as specified in DOM Features.

Method Detail

hasFeature

```
boolean hasFeature(java.lang.String feature,
                   java.lang.String version)
```

Test if the DOM implementation implements a specific feature and version, as specified in DOM Features.

Parameters:

`feature` - The name of the feature to test.

`version` - This is the version number of the feature to test.

Returns:

`true` if the feature is implemented in the specified version, `false` otherwise.

~~createDocumentType~~

```
DocumentType createDocumentType(java.lang.String qualifiedName,
                                java.lang.String publicId,
                                java.lang.String systemId)
                                throws DOMException
```

~~Creates an empty `DocumentType` node. Entity declarations and notations are not made available. Entity reference expansions and default attribute additions do not occur..~~

~~Parameters:~~

~~`qualifiedName` - The qualified name of the document type to be created.~~

~~`publicId` - The external subset public identifier.~~

~~`systemId` - The external subset system identifier.~~

~~Returns:~~

~~A new `DocumentType` node with `Node.ownerDocument` set to `null`.~~

~~Throws:~~

~~`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified qualified name is not an XML name according to [XML 1.0].~~

~~NAMESPACE_ERR: Raised if the `qualifiedName` is malformed.~~

NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature "XML" and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

createDocument

```
Document createDocument(java.lang.String namespaceURI,
                        java.lang.String qualifiedName,
                        DocumentType doctype)
                        throws DOMException
```

Creates a DOM Document object of the specified type with its document element. Note that based on the `DocumentType` given to create the document, the implementation may instantiate specialized `Document` objects that support additional features than the "Core", such as "HTML" [DOM Level 2 HTML] . On the other hand, setting the `DocumentType` after the document was created makes this very unlikely to happen. Alternatively, specialized `Document` creation methods, such as `createHTMLDocument` [DOM Level 2 HTML] , can be used to obtain specific types of `Document` objects.

Parameters:

`namespaceURI` - The namespace URI of the document element to create or `null`

`qualifiedName` - The qualified name of the document element to be created or `null`

`doctype` - The type of document to be created or `null`. When `doctype` is not `null`, its `Node.ownerDocument` attribute is set to the document being created.

Returns:

A new `Document` object with its document element. If the `NamespaceURI`, `qualifiedName`, and `doctype` are `null`, the returned `Document` is empty with no document element.

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified qualified name is not an XML name according to [XML 1.0].

NAMESPACE_ERR: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is `null`, or if the `qualifiedName` is `null` and the `namespaceURI` is different from `null`, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "

http://www.w3.org/XML/1998/namespace" [XML Namespaces] , or if the DOM implementation does not support the `"XML"` feature but a non-null namespace URI was provided, since namespaces were defined by XML.
WRONG_DOCUMENT_ERR: Raised if `doctype` has already been used with a different document or was created from a different implementation.
NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature "XML" and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

getFeature

```
java.lang.Object getFeature(java.lang.String feature,
                            java.lang.String version)
```

This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in DOM Features. The specialized object may also be obtained by using binding-specific casting methods but is not necessarily expected to, as discussed in . This method also allow the implementation to provide specialized objects which do not support the `DOMImplementation` interface.

Parameters:

`feature` - The name of the feature requested. Note that any plus sign "+" prepended to the name of the feature will be ignored since it is not significant in the context of this method.
`version` - This is the version number of the feature to test.

Returns:

Returns an object which implements the specialized APIs of the specified feature and version, if any, or `null` if there is no object which implements interfaces associated with that feature. If the `DOMObject` returned by this method implements the `DOMImplementation` interface, it must delegate to the primary core `DOMImplementation` and not return results inconsistent with the primary core `DOMImplementation` such as `hasFeature`, `getFeature`, etc.

Since:

DOM Level 3

---

17 - Oct - 08

org.w3c.dom

Interface DOMImplementationList

these are good design in ABSTRACTION

`public interface` **`DOMImplementationList`**

The `DOMImplementationList` interface provides the abstraction of an ordered collection of DOM implementations, without defining or constraining how this collection is implemented. The items in the `DOMImplementationList` are accessible via an integral index, starting from 0.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

`int`	**`getLength`**`()` The number of `DOMImplementation`s in the list.
`DOMImplementation`	**`item`**`(int index)` Returns the `index`th item in the collection.

Method Detail

item

`DOMImplementation` **`item`**`(int index)`

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of `DOMImplementation`s in the list, this returns `null`.

Parameters:

`index` - Index into the collection.

Returns:

The `DOMImplementation` at the `index` th position in the `DOMImplementationList`, or `null` if that is not a valid index.

---

getLength

`int` **`getLength`**`()`

The number of `DOMImplementation`s in the list. The range of valid child node indices is 0 to `length-1` inclusive.

---

24 - Oct - 08

22 - Dec - 08

it has 4 methods.

All are Factory methods to create
1. LSParser, LSSerializer
2. LSInput, LSOutput

org.w3c.dom.ls

```
domImplementationRegistry = DOMImplementationRegistry.newInstance();
   logger.info("domImplementationRegistry "+domImplementationRegistry);

   implementationLS = (DOMImplementationLS)domImplementationRegistry.getDOMImplementation("LS
3.0");
   logger.info("implementationLS ::"+implementationLS);

   lsParser = implementationLS.createLSParser(DOMImplementationLS.MODE_SYNCHRONOUS, XMLConstants.
W3C_XML_SCHEMA_NS_URI);
   logger.info("lsParser ::"+lsParser);
```

DOMImplementationLS contains the factory methods for creating Load and Save objects.

The expectation is that an instance of the `DOMImplementationLS` interface can be obtained by using binding-specific casting methods on an instance of the `DOMImplementation` interface or, if the `Document` supports the feature `"Core"` version `"3.0"` defined in [DOM Level 3 Core] , by using the method `DOMImplementation.getFeature` with parameter values `"LS"` (or `"LS-Async"`) and `"3.0"` (respectively).

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Field Summary

static short	**MODE_ASYNCHRONOUS** Create an asynchronous `LSParser`.
static short	**MODE_SYNCHRONOUS** Create a synchronous `LSParser`.

Method Summary

LSInput	**createLSInput**() Create a new empty input source object where LSInput.characterStream, LSInput.byteStream , LSInput.stringData LSInput.systemId, LSInput.publicId, LSInput.baseURI, and LSInput.encoding are null, and LSInput.certifiedText is false.
LSOutput	**createLSOutput**() Create a new empty output destination object where LSOutput.characterStream, LSOutput.byteStream, LSOutput.systemId, LSOutput.encoding are null.
LSParser	**createLSParser**(short mode, java.lang.String schemaType) Create a new LSParser.
LSSerializer	**createLSSerializer**() Create a new LSSerializer object.

Field Detail

MODE_SYNCHRONOUS

```
static final short MODE_SYNCHRONOUS
```

Create a synchronous LSParser.

this option only to parser

See Also:

Constant Field Values

MODE_ASYNCHRONOUS

```
static final short MODE_ASYNCHRONOUS
```

Create an asynchronous LSParser.

this option only to parser

See Also:

Constant Field Values

Method Detail

createLSParser

```
LSParser createLSParser(short mode,
                        java.lang.String schemaType)
                        throws DOMException
```

Create a new `LSParser`. The newly constructed parser may then be configured by means of its `DOMConfiguration` object, and used to parse documents by means of its `parse` method.

Parameters:

`mode` - The `mode` argument is either `MODE_SYNCHRONOUS` or `MODE_ASYNCHRONOUS`, if `mode` is `MODE_SYNCHRONOUS` then the `LSParser` that is created will operate in synchronous mode, if it's `MODE_ASYNCHRONOUS` then the `LSParser` that is created will operate in asynchronous mode.

`schemaType` - An absolute URI representing the type of the schema language used during the load of a `Document` using the newly created `LSParser`. Note that no lexical checking is done on the absolute URI. In order to create a `LSParser` for any kind of schema types (i.e. the LSParser will be free to use any schema found), use the value `null`.

Note: For W3C XML Schema [XML Schema Part 1] , applications must use the value `"http://www.w3.org/2001/XMLSchema"`. For XML DTD [XML 1.0], applications must use the value `"http://www.w3.org/TR/REC-xml"`. Other Schema languages are outside the scope of the W3C and therefore should recommend an absolute URI in order to use this method.

can it support SAX Error Handler ?

Returns:

The newly created `LSParser` object. This `LSParser` is either synchronous or asynchronous depending on the value of the `mode` argument.

Note: By default, the newly created `LSParser` does not contain a `DOMErrorHandler`, i.e. the value of the "error-handler" configuration parameter is `null`. However, implementations may provide a default error handler at creation time. In that case, the initial value of the `"error-handler"` configuration parameter on the new `LSParser` object contains a reference to the default error handler.

Throws:

DOMException - NOT_SUPPORTED_ERR: Raised if the requested mode or schema type is not supported.

---

createLSSerializer

LSSerializer **createLSSerializer**()

Create a new LSSerializer object.

Returns:

The newly created LSSerializer object.

Note: By default, the newly created LSSerializer has no DOMErrorHandler, i.e. the value of the "error-handler" configuration parameter is null. However, implementations may provide a default error handler at creation time. In that case, the initial value of the "error-handler" configuration parameter on the new LSSerializer object contains a reference to the default error handler.

---

createLSInput

LSInput **createLSInput**()

Create a new empty input source object where LSInput.characterStream, LSInput.byteStream, LSInput.stringData LSInput.systemId, LSInput.publicId, LSInput.baseURI, and LSInput.encoding are null, and LSInput.certifiedText is false.

Returns:

The newly created input object.

---

createLSOutput

LSOutput **createLSOutput**()

Create a new empty output destination object where `LSOutput.characterStream`, `LSOutput.byteStream`, `LSOutput.systemId`, `LSOutput.encoding` are null.

Returns:

The newly created output object.

---

24 - Oct - 08

it has 4 methods.
i used this class explicitly in LS module

org.w3c.dom.bootstrap

Class DOMImplementationRegistry

```
java.lang.Object
  └ org.w3c.dom.bootstrap.DOMImplementationRegistry
```

if you want to know what the different implementation of DOM available in current JAXP JAR file. use this class

```
public final class DOMImplementationRegistry
```

extends java.lang.Object

A factory that enables applications to obtain instances of `DOMImplementation`.

its main purpose programatic configuration of a another DOM implemention

Example:

```
// get an instance of the DOMImplementation registry
DOMImplementationRegistry registry =
     DOMImplementationRegistry.newInstance();
// get a DOM implementation the Level 3 XML module
DOMImplementation domImpl =
     registry.getDOMImplementation("XML 3.0");
```

registry.getDOMImplementation(" LS 3.0 ")

This provides an application with an implementation-independent starting point. DOM implementations may modify this class to meet new security standards or to provide *additional* fallbacks for the list of DOMImplementationSources.

Since:

DOM Level 3

See Also:

`DOMImplementation`, `DOMImplementationSource`

Field Summary

`static java.lang.String`	**PROPERTY** The system property to specify the DOMImplementationSource class names.

Method Summary

void	**addSource**(DOMImplementationSource s) Register an implementation.
DOMImplementation	**getDOMImplementation**(java.lang.String features) Return the first implementation that has the desired features, or null if none is found.
DOMImplementationList	**getDOMImplementationList**(java.lang.String features) Return a list of implementations that support the desired features.
static DOMImplementationRegistry	**newInstance**() Obtain a new instance of a DOMImplementationRegistry.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

PROPERTY

```
public static final java.lang.String PROPERTY
```

The system property to specify the DOMImplementationSource class names.

See Also:
Constant Field Values

Method Detail

newInstance

```
public static DOMImplementationRegistry newInstance()
                                                throws java.lang.ClassNotFoundException,
                                                       java.lang.InstantiationException,
                                                       java.lang.IllegalAccessException,
                                                       java.lang.ClassCastException
```

Obtain a new instance of a DOMImplementationRegistry. The DOMImplementationRegistry is initialized by the application or the implementation, depending on the context, by first checking the value of the Java system property org.w3c.dom.DOMImplementationSourceList and the the service provider whose contents are at "META_INF/services/org.w3c.dom.DOMImplementationSourceList" The value of this property is a white-space separated list of names of availables classes implementing the DOMImplementationSource interface. Each class listed in the class name list is instantiated and any exceptions encountered are thrown to the application.

Returns:
an initialized instance of DOMImplementationRegistry
Throws:
`java.lang.ClassNotFoundException` - If any specified class can not be found
`java.lang.InstantiationException` - If any specified class is an interface or abstract class
`java.lang.IllegalAccessException` - If the default constructor of a specified class is not accessible
`java.lang.ClassCastException` - If any specified class does not implement `DOMImplementationSource`

---

getDOMImplementation

```
public DOMImplementation getDOMImplementation(java.lang.String features)
```

Return the first implementation that has the desired features, or `null` if none is found.

Parameters:
`features` - A string that specifies which features are required. This is a space separated list in which each feature is specified by its name optionally followed by a space and a version number. This is something like: "XML 1.0 Traversal +Events 2.0"
Returns:
An implementation that has the desired features, or `null` if none found.

---

getDOMImplementationList

```
public DOMImplementationList getDOMImplementationList(java.lang.String features)
```

Return a list of implementations that support the desired features.

Parameters:
`features` - A string that specifies which features are required. This is a space separated list in which each feature is specified by its name optionally followed by a space and a version number. This is something like: "XML 1.0 Traversal +Events 2.0"
Returns:
A list of DOMImplementations that support the desired features.

---

addSource

```
public void addSource(DOMImplementationSource s)
```

Register an implementation.

Parameters:

s - The source to be registered, may not be `null`

---

org.w3c.dom

Interface DOMImplementationSource

`public interface` **`DOMImplementationSource`**

This interface permits a DOM implementer to supply one or more implementations, based upon requested features and versions, as specified in . Each implemented `DOMImplementationSource` object is listed in the binding-specific list of available sources so that its `DOMImplementation` objects are made available.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

DOMImplementation	**`getDOMImplementation`**`(java.lang.String features)` A method to request the first DOM implementation that supports the specified features.
DOMImplementationList	**`getDOMImplementationList`**`(java.lang.String features)` A method to request a list of DOM implementations that support the specified features and versions, as specified in .

Method Detail

getDOMImplementation

`DOMImplementation` **`getDOMImplementation`**`(java.lang.String features)`

A method to request the first DOM implementation that supports the specified features.

Parameters:

`features` - A string that specifies which features and versions are required. This is a space separated list in which each feature is specified by its name optionally followed by a space and a version number. This method returns the first item of the list returned by `getDOMImplementationList`. As an example, the string `"XML 3.0 Traversal +Events 2.0"` will request a DOM implementation that supports the module "XML" for its 3.0 version, a module that support of the "Traversal" module for any version, and the module "Events" for its 2.0 version. The module "Events" must be accessible using the method `Node.getFeature()` and `DOMImplementation.getFeature()`.

Returns:

The first DOM implementation that support the desired features, or `null` if this source has none.

---

getDOMImplementationList

`DOMImplementationList` **`getDOMImplementationList`**`(java.lang.String features)`

A method to request a list of DOM implementations that support the specified features and versions, as specified in .

Parameters:

`features` - A string that specifies which features and versions are required. This is a space separated list in which each feature is specified by its name optionally followed by a space and a version number. This is something like: "XML 3.0 Traversal +Events 2.0"

Returns:

A list of DOM implementations that support the desired features.

---

17 - Oct - 08

19 - Dec - 08

it has 6 methods

org.w3c.dom

Interface DOMLocator

i dont know how to execute. atleast 5th round find use net to get sample

```
public interface DOMLocator
```

`DOMLocator` is an interface that describes a location (e.g. where an error occurred).

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

int	**getByteOffset**() The byte offset into the input source this locator is pointing to or `-1` if there is no byte offset available.
int	**getColumnNumber**() The column number this locator is pointing to, or `-1` if there is no column number available.
int	**getLineNumber**() The line number this locator is pointing to, or `-1` if there is no column number available.
Node	**getRelatedNode**() The node this locator is pointing to, or `null` if no node is available.
java.lang.String	**getUri**() The URI this locator is pointing to, or `null` if no URI is available.

int	**getUtf16Offset()** The UTF-16, as defined in [Unicode] and Amendment 1 of [ISO/IEC 10646], offset into the input source this locator is pointing to or `-1` if there is no UTF-16 offset available.

Method Detail

getLineNumber

`int` **`getLineNumber`**`()`

The line number this locator is pointing to, or `-1` if there is no column number available.

---

getColumnNumber

`int` **`getColumnNumber`**`()`

The column number this locator is pointing to, or `-1` if there is no column number available.

---

getByteOffset

`int` **`getByteOffset`**`()`

The byte offset into the input source this locator is pointing to or `-1` if there is no byte offset available.

---

getUtf16Offset

`int` **`getUtf16Offset`**`()`

The UTF-16, as defined in [Unicode] and Amendment 1 of [ISO/IEC 10646], offset into the input source this locator is pointing to or `-1` if there is no UTF-16 offset available.

---

getRelatedNode

`Node` **`getRelatedNode`**`()`

The node this locator is pointing to, or `null` if no node is available.

---

getUri

`java.lang.String` **`getUri`**`()`

The URI this locator is pointing to, or `null` if no URI is available.

---

17 - Oct - 08

it is used only one DOMConfiguration.
So, no need know as of now

org.w3c.dom

Interface DOMStringList

`public interface` **`DOMStringList`**

The `DOMStringList` interface provides the abstraction of an ordered collection of `DOMString` values, without defining or constraining how this collection is implemented. The items in the `DOMStringList` are accessible via an integral index, starting from 0.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

`boolean`	**`contains`**`(java.lang.String str)` Test if a string is part of this `DOMStringList`.
`int`	**`getLength`**`()` The number of `DOMString`s in the list.
`java.lang.String`	**`item`**`(int index)` Returns the `index`th item in the collection.

Method Detail

item

`java.lang.String` **`item`**`(int index)`

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of `DOMString`s in the list, this returns `null`.

Parameters:

`index` - Index into the collection.

Returns:

The `DOMString` at the `index`th position in the `DOMStringList`, or `null` if that is not a valid index.

---

getLength

`int` **`getLength`**`()`

The number of `DOMString`s in the list. The range of valid child node indices is 0 to `length-1` inclusive.

---

contains

`boolean` **`contains`**`(java.lang.String str)`

Test if a string is part of this `DOMStringList`.

Parameters:

`str` - The string to look for.

Returns:

`true` if the string has been found, `false` otherwise.

---

17 - Dec - 08

org.w3c.dom

Interface Element

16 - methods related to Attribute
2 - methods for child
1 - schema type
1 - get name of element

All Superin

Node

NameSpace(NS) methods are not hands on still....

i enabled namespace feature in Factory instance too..

try to familiar in 5th round at least

totally 20 methods.

mostly methods are overloaded by string, name space, Attr node

```
public interface Element
```

extends Node

The `Element` interface represents an element in an HTML or XML document. Elements may have attributes associated with them; since the `Element` interface inherits from `Node`, the generic `Node` interface attribute `attributes` may be used to retrieve the set of all attributes for an element. There are methods on the `Element` interface to retrieve either an `Attr` object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an `Attr` object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience.

Note: In DOM Level 2, the method `normalize` is inherited from the `Node` interface where it was moved.

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

java.lang.String	**getAttribute**(java.lang.String name) Retrieves an attribute value by name.
Attr	**getAttributeNode**(java.lang.String name) Retrieves an attribute node by name.
Attr	**getAttributeNodeNS**(java.lang.String namespaceURI, java.lang.String localName) Retrieves an Attr node by local name and namespace URI.
java.lang.String	**getAttributeNS**(java.lang.String namespaceURI, java.lang.String localName) Retrieves an attribute value by local name and namespace URI.
NodeList	**getElementsByTagName**(java.lang.String name) Returns a NodeList of all descendant Elements with a given tag name, in document order.
NodeList	**getElementsByTagNameNS**(java.lang.String namespaceURI, java.lang.String localName) Returns a NodeList of all the descendant Elements with a given local name and namespace URI in document order.
TypeInfo	**getSchemaTypeInfo**() The type information associated with this element.
java.lang.String	**getTagName**() The name of the element.

only document methods take aName as parameter. rest of them take localName

this method would useful only, if the descendant element has its own namespace

boolean	**hasAttribute**(java.lang.String name) Returns `true` when an attribute with a given name is specified on this element or has a default value, `false` otherwise.
boolean	**hasAttributeNS**(java.lang.String namespaceURI, java.lang.String localName) Returns `true` when an attribute with a given local name and namespace URI is specified on this element or has a default value, `false` otherwise.
void	**removeAttribute**(java.lang.String name) Removes an attribute by name.
Attr	**removeAttributeNode**(Attr oldAttr) Removes the specified attribute node.
void	**removeAttributeNS**(java.lang.String namespaceURI, java.lang.String localName) Removes an attribute by local name and namespace URI.
void	**setAttribute**(java.lang.String name, java.lang.String value) Adds a new attribute.
Attr	**setAttributeNode**(Attr newAttr) Adds a new attribute node.
Attr	**setAttributeNodeNS**(Attr newAttr) Adds a new attribute.
void	**setAttributeNS**(java.lang.String namespaceURI, java.lang.String qualifiedName, java.lang.String value) Adds a new attribute.
void	**setIdAttribute**(java.lang.String name, boolean isId) If the parameter `isId` is `true`, this method declares the specified attribute to be a user-determined ID attribute .
void	**setIdAttributeNode**(Attr idAttr, boolean isId) If the parameter `isId` is `true`, this method declares the specified attribute to be a user-determined ID attribute .
void	**setIdAttributeNS**(java.lang.String namespaceURI, java.lang.String localName, boolean isId) If the parameter `isId` is `true`, this method declares the specified attribute to be a user-determined ID attribute .

not node

No Special getter methods for these.

can any 'arbitrary' name be as ID attribute instead of 'id' ?????????
ANS: yes. this is what actual purpose these methods

Methods inherited from interface org.w3c.dom.Node
appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

Method Detail

getTagName

`java.lang.String` **`getTagName`**`()`

The name of the element. If `Node.localName` is different from `null`, this attribute is a qualified name. For example, in:

```
<elementExample id="demo"> ...
</elementExample> ,
```

`tagName` has the value `"elementExample"`. Note that this is case-preserving in XML, as are all of the operations of the DOM. The HTML DOM returns the `tagName` of an HTML element in the canonical uppercase form, regardless of the case in the source HTML document.

---

getAttribute

`java.lang.String` **`getAttribute`**`(java.lang.String name)`

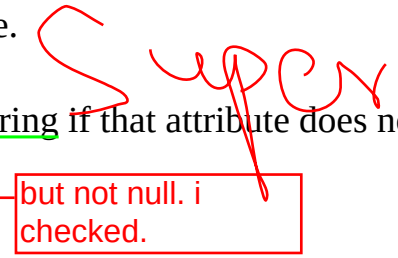
Retrieves an attribute value by name.

Parameters:

`name` - The name of the attribute to retrieve.

Returns:

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



---

setAttribute

```
void setAttribute(java.lang.String name,
                  java.lang.String value)
                  throws DOMException
```

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. So 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.

To set an attribute with a qualified name and namespace URI, use the `setAttributeNS` method.

Parameters:

`name` - The name of the attribute to create or alter.

`value` - Value to set in string form.

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified name is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

---

removeAttribute

```
void removeAttribute(java.lang.String name)
                     throws DOMException
```

Removes an attribute by name. If a default value for the removed attribute is defined in the DTD,

a new attribute immediately appears with the default value as well as the corresponding namespace URI, local name, and prefix when applicable. The implementation may handle default values from other schemas similarly but applications should use `Document.normalizeDocument()` to guarantee this information is up-to-date.
If no attribute with this name is found, this method has no effect.
To remove an attribute by local name and namespace URI, use the `removeAttributeNS` method.

Parameters:

`name` - The name of the attribute to remove.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

---

getAttributeNode

`Attr` **`getAttributeNode`**`(java.lang.String name)`

Retrieves an attribute node by name.
~~To retrieve an attribute node by qualified name and namespace URI, use the~~ `getAttributeNodeNS` ~~method.~~

Parameters:

`name` - The name (`nodeName`) of the attribute to retrieve.

Returns:

The `Attr` node with the specified name (`nodeName`) or `null` if there is no such attribute.

but, that getAttribute(name) will give empty string

---

setAttributeNode

`Attr` **`setAttributeNode`**`(Attr newAttr)`
`throws DOMException`

Adds a new attribute node. If an attribute with that name (`nodeName`) is already present in the element, it is replaced by the new one. Replacing an attribute node by itself has no effect.
~~To add a new attribute node with a qualified name and namespace URI, use the~~

`setAttributeNodeNS` method.

if return null, then it is new attribute (i tested)

Parameters:

`newAttr` - The `Attr` node to add to the attribute list.

Returns:

If the `newAttr` attribute replaces an existing attribute, the replaced `Attr` node is returned, otherwise `null` is returned.

Throws:

`DOMException` - WRONG_DOCUMENT_ERR: Raised if `newAttr` was created from a different document than the one that created the element.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
INUSE_ATTRIBUTE_ERR: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.

---

removeAttributeNode

`Attr` **`removeAttributeNode`**`(Attr oldAttr)`
`throws DOMException`

Removes the specified attribute node. If a default value for the removed `Attr` node is defined in the DTD, a new node immediately appears with the default value as well as the corresponding namespace URI, local name, and prefix when applicable. The implementation may handle default values from other schemas similarly but applications should use `Document.normalizeDocument()` to guarantee this information is up-to-date.

Parameters:

`oldAttr` - The `Attr` node to remove from the attribute list.

Returns:

The `Attr` node that was removed.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
NOT_FOUND_ERR: Raised if `oldAttr` is not an attribute of the element.

---

getElementsByTagName

`NodeList` **`getElementsByTagName`**`(java.lang.String name)`

Returns a `NodeList` of all descendant `Elements` with a given tag name, in document order.

Parameters:

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

Returns:

A list of matching `Element` nodes.

---

getAttributeNS

```
java.lang.String getAttributeNS(java.lang.String namespaceURI,
                                 java.lang.String localName)
                                 throws DOMException
```

Retrieves an attribute value by local name and namespace URI.
Per [XML Namespaces] , applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the attribute to retrieve.

`localName` - The local name of the attribute to retrieve.

Returns:

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

Throws:

`DOMException` - NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature `"XML"` and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

setAttributeNS

```
void setAttributeNS(java.lang.String namespaceURI,
                    java.lang.String qualifiedName,
                    java.lang.String value)
                    throws DOMException
```

Adds a new attribute. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the `qualifiedName`, and its value is changed to be the `value` parameter. This value is a simple string; it is not parsed as it is being set. So 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 `setAttributeNodeNS` or `setAttributeNode` to assign it as the value of an attribute. Per [XML Namespaces], applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the attribute to create or alter.
`qualifiedName` - The qualified name of the attribute to create or alter.
`value` - The value to set in string form.

Throws:

`DOMException` - INVALID_CHARACTER_ERR: Raised if the specified qualified name is not an XML name according to the XML version in use specified in the `Document.xmlVersion` attribute.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
NAMESPACE_ERR: Raised if the `qualifiedName` is malformed per the Namespaces in XML specification, if the `qualifiedName` has a prefix and the `namespaceURI` is `null`, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from " http://www.w3.org/XML/1998/namespace", if the `qualifiedName` or its prefix is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/", or if the `namespaceURI` is "http://www.w3.org/2000/xmlns/" and neither the `qualifiedName` nor its prefix is "xmlns".
NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature `"XML"` and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

removeAttributeNS

```
void removeAttributeNS(java.lang.String namespaceURI,
                       java.lang.String localName)
                       throws DOMException
```

Removes an attribute by local name and namespace URI. If a default value for the removed attribute is defined in the DTD, a new attribute immediately appears with the default value as well as the corresponding namespace URI, local name, and prefix when applicable. The implementation may handle default values from other schemas similarly but applications should use `Document.normalizeDocument()` to guarantee this information is up-to-date.
If no attribute with this local name and namespace URI is found, this method has no effect.
Per [XML Namespaces] , applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the attribute to remove.

`localName` - The local name of the attribute to remove.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature `"XML"` and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

getAttributeNodeNS

```
Attr getAttributeNodeNS(java.lang.String namespaceURI,
                        java.lang.String localName)
                        throws DOMException
```

Retrieves an `Attr` node by local name and namespace URI.
Per [XML Namespaces] , applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the attribute to retrieve.
`localName` - The local name of the attribute to retrieve.

Returns:

The `Attr` node with the specified attribute local name and namespace URI or `null` if there is no such attribute.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature `"XML"` and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

this can be a attribute object without namespace and prefix. (i tested)

this cannot be null

setAttributeN

```
Attr setAttributeNodeNS(Attr newAttr)
                        throws DOMException
```

Adds a new attribute. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one. Replacing an attribute node by itself has no effect.
Per [XML Namespaces] , applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`newAttr` - The `Attr` node to add to the attribute list.

Returns:

If the `newAttr` attribute replaces an existing attribute with the same local name and namespace URI, the replaced `Attr` node is returned, otherwise `null` is returned.

Throws:

`DOMException` - WRONG_DOCUMENT_ERR: Raised if `newAttr` was created from a different document than the one that created the element.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
INUSE_ATTRIBUTE_ERR: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.
~~NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature `"XML"` and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).~~

Since:

DOM Level 2

---

getElementsByTagNameNS

```
NodeList getElementsByTagNameNS(java.lang.String namespaceURI,
                                java.lang.String localName)
                                throws DOMException
```

Returns a `NodeList` of all the descendant `Elements` with a given local name and namespace URI in document order.

Parameters:

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

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

Returns:

A new `NodeList` object containing all the matched `Elements`.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature `"XML"` and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

hasAttribute

```
boolean hasAttribute(java.lang.String name)
```

Returns `true` when an attribute with a given name is specified on this element or has a default value, `false` otherwise.

Parameters:

`name` - The name of the attribute to look for.

Returns:

`true` if an attribute with the given name is specified on this element or has a default value, `false` otherwise.

Since:

DOM Level 2

---

hasAttributeNS

```
boolean hasAttributeNS(java.lang.String namespaceURI,
                       java.lang.String localName)
                       throws DOMException
```

can i use here * as argument ?.
ANS.. no, it does not work (i tested)

both cannot be as *

Returns `true` when an attribute with a given local name and namespace URI is specified on this element or has a default value, `false` otherwise.
Per [XML Namespaces] , applications must use the value `null` as the `namespaceURI` parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the attribute to look for.
`localName` - The local name of the attribute to look for.

Returns:

`true` if an attribute with the given local name and namespace URI is specified or has a default value on this element, `false` otherwise.

Throws:

DOMException - NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature "XML" and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

getSchemaTypeInfo

```
TypeInfo getSchemaTypeInfo()
```

The type information associated with this element.

Since:

DOM Level 3

---

setIdAttribute

this is the only way in JAXP to set a ID (for uniqeness) by programattically

we can make any ONLY EXISTING attribute as id for this element. (i tested)

```
void setIdAttribute(java.lang.String name,
                    boolean isId)
                    throws DOMException
```

If the parameter `isId` is `true`, this method declares the specified attribute to be a user-determined ID attribute . This affects the value of `Attr.isId` and the behavior of `Document.getElementById`, but does not change any schema that may be in use, in particular this does not affect the `Attr.schemaTypeInfo` of the specified `Attr` node. Use the value `false` for the parameter `isId` to undeclare an attribute for being a user-determined ID attribute.
To specify an attribute by local name and namespace URI, use the `setIdAttributeNS` method.

Parameters:

`name` - The name of the attribute.

`isId` - Whether the attribute is a of type ID.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
NOT_FOUND_ERR: Raised if the specified node is not an attribute of this element.

Since:

DOM Level 3

---

setIdAttributeNS

```
void setIdAttributeNS(java.lang.String namespaceURI,
                      java.lang.String localName,
                      boolean isId)
                      throws DOMException
```

If the parameter `isId` is `true`, this method declares the specified attribute to be a user-determined ID attribute . This affects the value of `Attr.isId` and the behavior of `Document.getElementById`, but does not change any schema that may be in use, in particular this does

not affect the `Attr.schemaTypeInfo` of the specified `Attr` node. Use the value `false` for the parameter `isId` to undeclare an attribute for being a user-determined ID attribute.

Parameters:

`namespaceURI` - The namespace URI of the attribute.

`localName` - The local name of the attribute.

`isId` - Whether the attribute is a of type ID.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if the specified node is not an attribute of this element.

Since:

DOM Level 3

---

setIdAttributeNode

```
void setIdAttributeNode(Attr idAttr,
                        boolean isId)
                        throws DOMException
```

If the parameter `isId` is `true`, this method declares the specified attribute to be a user-determined ID attribute . This affects the value of `Attr.isId` and the behavior of `Document.getElementById`, but does not change any schema that may be in use, in particular this does not affect the `Attr.schemaTypeInfo` of the specified `Attr` node. Use the value `false` for the parameter `isId` to undeclare an attribute for being a user-determined ID attribute.

Parameters:

`idAttr` - The attribute node.

`isId` - Whether the attribute is a of type ID.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if the specified node is not an attribute of this element.

Since:

DOM Level 3

Package org.w3c.dom

20 - Interface

we can parse. but we cannot save the xml.
To save has to use DOM LS API

Interface Summary

Attr	The `At` `ent` object.
CDATASection	CDAT `taining` charact `).`
CharacterData	The `CharacterData` interface extends Node with a set of attributes and methods for accessing character data in the DOM.
Comment	This interface inherits from `CharacterData` and represents the content of a comment, i.e., all the characters between the starting '`<!`
Document	The `Document` interface represents the entire HTML or XML document.
DocumentFragment	`DocumentFragment` is a "lightweight" or "minimal" `Document` object.
~~DocumentType~~	~~Each `Document` has a `doctype` attribute whose value is either `null` or a `DocumentType` object.~~
DOMConfiguration	The `DOMConfiguration` interface represents the configuration of a document and maintains a table of recognized parameters.
DOMError	`DOMError` is an interface that describes an error.
DOMErrorHandler	`DOMErrorHandler` is a callback interface that the DOM implementation can call when reporting errors that happens while processing XML data, or when doing some other processing (e.g. validating a document). can't i use SAX's ErrorHandler ??
DOMImplementation	The `DOMImplementation` interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.
DOMImplementationList	The `DOMImplementationList` interface provides the abstraction of an ordered collection of DOM implementations, without defining or constraining how this collection is implemented.

DOMImplementationSource	This interface permits a DOM implementer to supply one or more implementations, based upon requested features and versions, as specified in .
DOMLocator	`DOMLocator` is an interface that describes a location (e.g.
DOMStringList	The `DOMStringList` interface provides the abstraction of an ordered collection of `DOMString` values, without defining or constraining how this collection is implemented.
Element	The `Element` interface represents an element in an HTML or XML document.
~~**Entity**~~	~~This interface represents a known entity, either parsed or unparsed, in an XML document.~~
~~**EntityReference**~~	~~`EntityReference` nodes may be used to represent an entity reference in the tree.~~
NamedNodeMap	Objects implementing the `NamedNodeMap` interface are used to represent collections of nodes that can be accessed by name.
NameList	The `NameList` interface provides the abstraction of an ordered collection of parallel pairs of name and namespace values (which could be null values), without defining or constraining how this collection is implemented.
Node	The `Node` interface is the primary datatype for the entire Document Object Model.
NodeList	The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.
~~**Notation**~~	~~This interface represents a notation declared in the DTD.~~
ProcessingInstruction	The `ProcessingInstruction` interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.
Text	The `Text` interface inherits from `CharacterData` and represents the textual content (termed character data in XML) of an `Element` or `Attr`.
TypeInfo	The `TypeInfo` interface represents a type referenced from `Element` or `Attr` nodes, specified in the schemas associated with the document.

UserDataHandler	When associating an object to a key on a node using `Node.setUserData()` the application can provide a handler that gets called when the node the object is associated to is being cloned, imported, or renamed.

Exception Summary

DOMException	DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable).

Package org.w3c.dom.bootstrap

i used this package in Load and Save (LS) module of DOM

Class Summary

DOMImplementationRegistry	A factory that enables applications to obtain instances of DOMImplementation.

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES All Classes

it is alternative for core DOM Api and Traversal API, but limited feature. i guess

Package org.w3c.dom.ls

it has been added in DOM - 3

24 - Oct - 08

Interface Summary

DOMImplementationLS	`DOMImplementationLS` contains the factory methods for creating Load and Save objects.
LSInput	This interface represents an input source for data.
LSLoadEvent	This interface represents a load event object that signals the completion of a document load.
LSOutput	This interface represents an output destination for data.
LSParser	An interface to an object that is able to build, or augment, a DOM tree from various input sources.
LSParserFilter	`LSParserFilter`s provide applications the ability to examine nodes as they are being constructed while parsing.
LSProgressEvent	This interface represents a progress event object that notifies the application about progress as a document is parsed.
LSResourceResolver	`LSResourceResolver` provides a way for applications to redirect references to external resources.
LSSerializer	A `LSSerializer` provides an API for serializing (writing) a DOM document out into XML.
LSSerializerFilter	`LSSerializerFilter`s provide applications the ability to examine nodes as they are being serialized and decide what nodes should be serialized or not.

Exception Summary

LSException	Parser or write operations may throw an `LSException` if the processing is stopped.

Package org.w3c.

it is, selected set of nodes. it mostly used with HTML. so, i deleted these class api.

Interface Summary

it will never use in JAXP (java) programming

DocumentRange	See also the Docum... Range Specification.
Range	See also the Docum... Range Specification.

Exception Summary

RangeException	Range operations may throw a `RangeException` as specified in their method descriptions.

21 - Oct - 08

Package org.w3c.dom.traversal

it has been added in DOM 2

this is one of the way or API used to walk through DOM Tree Nodes

Interface Summary

DocumentTraversal	`DocumentTraversal` contains methods that create `NodeIterators` and `TreeWalkers` to traverse a node and its children in document order (depth first, order in which the start tags occu
NodeFilter	Filte les.
NodeIterator	`NodeIterators` are used to step through a set of nodes, e.g.
TreeWalker	`TreeWalker` objects are used to navigate a document tree or subtree using the view of the document defined by their `whatToShow` flags and filter (if any).

This API is simply used to work with Document TREE . But not for parsing or save the XML

24 - Oct - 08

24 - Dec - 08

it has 16 methods.
All are simply Setter / getter methods

GOOD Programing

it has a method to set String of data instead of any file / stream / reader for xml parsing..

org.w3c.dom.ls

Interface LSInput

we can assume this class alternative for InputSource. nothing special

`public interface LSInput`

~~This interface represents an input source for data~~

~~This interface allows an application to encapsulate information about an input source in a single object, which may include a public identifier, a system identifier, a byte stream (possibly with a specified encoding), a base URI, and/or a character stream.~~

~~The exact definitions of a byte stream and a character stream are binding dependent.~~

~~The application is expected to provide objects that implement this interface whenever such objects are needed. The application can either provide its own objects that implement this interface, or it can use the generic factory method~~ `DOMImplementationLS.createLSInput()` ~~to create objects that implement this interface.~~

The `LSParser` will use the `LSInput` object to determine how to read data. The `LSParser` will look at the different inputs specified in the `LSInput` in the following order to know which one to read from, the first one that is not null and not an empty string will be used:

1. `LSInput.characterStream`
2. `LSInput.byteStream`
3. `LSInput.stringData`
4. `LSInput.systemId`
5. `LSInput.publicId`

If all inputs are null, the `LSParser` will report a `DOMError` with its `DOMError.type` set to `"no-input-specified"` and its `DOMError.severity` set to `DOMError.SEVERITY_FATAL_ERROR`.

yes. i got

`LSInput` objects belong to the application. The DOM implementation will never modify them (though it may make copies and modify the copies, if necessary).

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Method Summary

`java.lang.String`	**`getBaseURI`**`()` The base URI to be used (see section 5.1.4 in [IETF RFC 2396]) for resolving a relative `systemId` to an absolute URI.
`java.io.InputStream`	**`getByteStream`**`()` An attribute of a language and binding dependent type that represents a stream of bytes.
`boolean`	**`getCertifiedText`**`()` If set to true, assume that the input is certified (see section 2.13 in [XML 1.1]) when parsing [XML 1.1].
`java.io.Reader`	**`getCharacterStream`**`()` An attribute of a language and binding dependent type that represents a stream of 16-bit units.
`java.lang.String`	**`getEncoding`**`()` The character encoding, if known.
`java.lang.String`	**`getPublicId`**`()` The public identifier for this input source.
`java.lang.String`	**`getStringData`**`()` ← super method String data to parse.
`java.lang.String`	**`getSystemId`**`()` The system identifier, a URI reference [IETF RFC 2396], for this input source.
`void`	**`setBaseURI`**`(java.lang.String baseURI)` The base URI to be used (see section 5.1.4 in [IETF RFC 2396]) for resolving a relative `systemId` to an absolute URI.

void	**setByteStream**(java.io.InputStream byteStream) An attribute of a language and binding dependent type that represents a stream of bytes.
void	**setCertifiedText**(boolean certifiedText) If set to true, assume that the input is certified (see section 2.13 in [XML 1.1]) when parsing [XML 1.1].
void	**setCharacterStream**(java.io.Reader characterStream) An attribute of a language and binding dependent type that represents a stream of 16-bit units.
void	**setEncoding**(java.lang.String encoding) The character encoding, if known.
void	**setPublicId**(java.lang.String publicId) The public identifier for this input source.
void	**setStringData**(java.lang.String stringData) String data to parse.
void	**setSystemId**(java.lang.String systemId) The system identifier, a URI reference [IETF RFC 2396], for this input source.

Method Detail

getCharacterStream

`java.io.Reader` **`getCharacterStream`**`()`

An attribute of a language and binding dependent type that represents a stream of 16-bit units. The application must encode the stream using UTF-16 (defined in [Unicode] and in [ISO/IEC 10646]). It is not a requirement to have an XML declaration when using character streams. If an XML declaration is present, the value of the encoding attribute will be ignored.

---

setCharacterStream

`void` **`setCharacterStream`**`(java.io.Reader characterStream)`

An attribute of a language and binding dependent type that represents a stream of 16-bit units. The application must encode the stream using UTF-16 (defined in [Unicode] and in [ISO/IEC 10646]). It is not a requirement to have an XML declaration when using character streams. If an XML declaration is present, the value of the encoding attribute will be ignored.

---

getByteStream

`java.io.InputStream` **`getByteStream`**`()`

An attribute of a language and binding dependent type that represents a stream of bytes.
If the application knows the character encoding of the byte stream, it should set the encoding attribute. Setting the encoding in this way will override any encoding specified in an XML declaration in the data.

---

setByteStream

`void` **`setByteStream`**`(java.io.InputStream byteStream)`

An attribute of a language and binding dependent type that represents a stream of bytes.
If the application knows the character encoding of the byte stream, it should set the encoding attribute. Setting the encoding in this way will override any encoding specified in an XML declaration in the data.

---

getStringData

`java.lang.String` **`getStringData`**`()`

String data to parse. If provided, this will always be treated as a sequence of 16-bit units (UTF-16 encoded characters). It is not a requirement to have an XML declaration when using `stringData`. If an XML declaration is present, the value of the encoding attribute will be ignored.

---

setStringData

```
void setStringData(java.lang.String stringData)
```

String data to parse. If provided, this will always be treated as a sequence of 16-bit units (UTF-16 encoded characters). It is not a requirement to have an XML declaration when using `stringData`. If an XML declaration is present, the value of the encoding attribute will be ignored.

---

getSystemId

```
java.lang.String getSystemId()
```

The system identifier, a URI reference [IETF RFC 2396], for this input source. The system identifier is optional if there is a byte stream, a character stream, or string data. It is still useful to provide one, since the application will use it to resolve any relative URIs and can include it in error messages and warnings. (The LSParser will only attempt to fetch the resource identified by the URI reference if there is no other input available in the input source.)
If the application knows the character encoding of the object pointed to by the system identifier, it can set the encoding using the `encoding` attribute.
If the specified system ID is a relative URI reference (see section 5 in [IETF RFC 2396]), the DOM implementation will attempt to resolve the relative URI with the `baseURI` as the base, if that fails, the behavior is implementation dependent.

---

setSystemId

```
void setSystemId(java.lang.String systemId)
```

The system identifier, a URI reference [IETF RFC 2396], for this input source. The system identifier is optional if there is a byte stream, a character stream, or string data. It is still useful to provide one, since the application will use it to resolve any relative URIs and can include it in error messages and warnings. (The LSParser will only attempt to fetch the resource identified by the URI reference if there is no other input available in the input source.)
If the application knows the character encoding of the object pointed to by the system identifier, it can set the encoding using the `encoding` attribute.
If the specified system ID is a relative URI reference (see section 5 in [IETF RFC 2396]), the

DOM implementation will attempt to resolve the relative URI with the `baseURI` as the base, if that fails, the behavior is implementation dependent.

---

getPublicId

`java.lang.String` **`getPublicId`**`()`

The public identifier for this input source. This may be mapped to an input source using an implementation dependent mechanism (such as catalogues or other mappings). The public identifier, if specified, may also be reported as part of the location information when errors are reported.

---

setPublicId

`void` **`setPublicId`**`(java.lang.String publicId)`

The public identifier for this input source. This may be mapped to an input source using an implementation dependent mechanism (such as catalogues or other mappings). The public identifier, if specified, may also be reported as part of the location information when errors are reported.

---

getBaseURI

`java.lang.String` **`getBaseURI`**`()`

The base URI to be used (see section 5.1.4 in [IETF RFC 2396]) for resolving a relative `systemId` to an absolute URI.
If, when used, the base URI is itself a relative URI, an empty string, or null, the behavior is implementation dependent.

---

setBaseURI

`void` **`setBaseURI`**`(java.lang.String baseURI)`

The base URI to be used (see section 5.1.4 in [IETF RFC 2396]) for resolving a relative `systemId` to an absolute URI.
If, when used, the base URI is itself a relative URI, an empty string, or null, the behavior is implementation dependent.

---

getEncoding

`java.lang.String` **`getEncoding`**`()`

The character encoding, if known. The encoding must be a string acceptable for an XML encoding declaration ([XML 1.0] section 4.3.3 "Character Encoding in Entities").
This attribute has no effect when the application provides a character stream or string data. For other sources of input, an encoding specified by means of this attribute will override any encoding specified in the XML declaration or the Text declaration, or an encoding obtained from a higher level protocol, such as HTTP [IETF RFC 2616].

---

setEncoding

`void` **`setEncoding`**`(java.lang.String encoding)`

The character encoding, if known. The encoding must be a string acceptable for an XML encoding declaration ([XML 1.0] section 4.3.3 "Character Encoding in Entities").
This attribute has no effect when the application provides a character stream or string data. For other sources of input, an encoding specified by means of this attribute will override any encoding specified in the XML declaration or the Text declaration, or an encoding obtained from a higher level protocol, such as HTTP [IETF RFC 2616].

---

getCertifiedText

`boolean` **`getCertifiedText`**`()`

If set to true, assume that the input is certified (see section 2.13 in [XML 1.1]) when parsing

[XML 1.1].

---

setCertifiedText

```
void setCertifiedText(boolean certifiedText)
```

If set to true, assume that the input is certified (see section 2.13 in [XML 1.1]) when parsing [XML 1.1].

---

24 - Oct - 08

it has only 2 methods

org.w3c.dom.ls

Interface LSLoadEvent

All Superinterfaces:

Event

it is used only asynchronous LSParser only

this event can be evaluated only if parser is in ASYNCHRONOUS mode.

unfortunatly, the existing JAXP support only SYNCHRONOUS mode

IN 5th Round at least get to know how to run in ASync mode

```
public inter
```

extends Event

This interface represents a load event object that signals the completion of a document load.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Field Summary

Fields inherited from interface org.w3c.dom.events.Event
AT_TARGET, BUBBLING_PHASE, CAPTURING_PHASE

Method Summary

LSInput	**getInput**() The input source that was parsed.
Document	**getNewDocument**() The document that finished loading.

Methods inherited from interface org.w3c.dom.events.Event

getBubbles, getCancelable, getCurrentTarget, getEventPhase, getTarget, getTimeStamp, getType, initEvent, preventDefault, stopPropagation

Method Detail

getNewDocument

Document **getNewDocument**()

The document that finished loading.

---

getInput

LSInput **getInput**()

The input source that was parsed.

---

24 - Oct - 08

25 - Dec - 08

it has only 8 methods.
All are simply setter and getter methods

org.w3c.dom.ls

Interface LSOutput

public interface **LSOutput**

~~This interface represents an output destination for data.~~

~~This interface allows an application to encapsulate information about an output destination in a single object, which may include a URI, a byte stream (possibly with a specified encoding), a base URI, and/or a character stream.~~

~~The exact definitions of a byte stream and a character stream are binding dependent.~~

~~The application is expected to provide objects that implement this interface whenever such objects are needed. The application can either provide its own objects that implement this interface, or it can use the generic factory method `DOMImplementationLS.createLSOutput()` to create objects that implement this interface.~~

The `LSSerializer` will use the `LSOutput` object to determine where to serialize the output to. The `LSSerializer` will look at the different outputs specified in the `LSOutput` in the following order to know which one to output to, the first one that is not null and not an empty string will be used:

1. `LSOutput.characterStream`
2. `LSOutput.byteStream`
3. `LSOutput.systemId`

`LSOutput` objects belong to the application. The DOM implementation will never modify them (though it may make copies and modify the copies, if necessary).

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Method Summary

java.io.OutputStream	**getByteStream**() An attribute of a language and binding dependent type that represents a writable stream of bytes.
java.io.Writer	**getCharacterStream**() An attribute of a language and binding dependent type that represents a writable stream to which 16-bit units can be output.
java.lang.String	**getEncoding**() The character encoding to use for the output.
java.lang.String	**getSystemId**() The system identifier, a URI reference [IETF RFC 2396], for this output destination.
void	**setByteStream**(java.io.OutputStream byteStream) An attribute of a language and binding dependent type that represents a writable stream of bytes.
void	**setCharacterStream**(java.io.Writer characterStream) An attribute of a language and binding dependent type that represents a writable stream to which 16-bit units can be output.
void	**setEncoding**(java.lang.String encoding) The character encoding to use for the output.
void	**setSystemId**(java.lang.String systemId) The system identifier, a URI reference [IETF RFC 2396], for this output destination.

Method Detail

getCharacterStream

```
java.io.Writer getCharacterStream()
```

An attribute of a language and binding dependent type that represents a writable stream to which 16-bit units can be output.

---

setCharacterStream

```
void setCharacterStream(java.io.Writer characterStream)
```

An attribute of a language and binding dependent type that represents a writable stream to which 16-bit units can be output.

---

getByteStream

```
java.io.OutputStream getByteStream()
```

An attribute of a language and binding dependent type that represents a writable stream of bytes.

---

setByteStream

```
void setByteStream(java.io.OutputStream byteStream)
```

An attribute of a language and binding dependent type that represents a writable stream of bytes.

---

getSystemId

```
java.lang.String getSystemId()
```

The system identifier, a URI reference [IETF RFC 2396], for this output destination.
If the system ID is a relative URI reference (see section 5 in [IETF RFC 2396]), the behavior is implementation dependent.

---

setSystemId

```
void setSystemId(java.lang.String systemId)
```

The system identifier, a URI reference [IETF RFC 2396], for this output destination.
If the system ID is a relative URI reference (see section 5 in [IETF RFC 2396]), the behavior is implementation dependent.

---

getEncoding

`java.lang.String` **`getEncoding`**`()`

The character encoding to use for the output. The encoding must be a string acceptable for an XML encoding declaration ([XML 1.0] section 4.3.3 "Character Encoding in Entities"), it is recommended that character encodings registered (as charsets) with the Internet Assigned Numbers Authority [IANA-CHARSETS] should be referred to using their registered names.

---

setEncoding

`void` **`setEncoding`**`(java.lang.String encoding)`

The character encoding to use for the output. The encoding must be a string acceptable for an XML encoding declaration ([XML 1.0] section 4.3.3 "Character Encoding in Entities"), it is recommended that character encodings registered (as charsets) with the Internet Assigned Numbers Authority [IANA-CHARSETS] should be referred to using their registered names.

---

24 - Dec - 08

it has 9 methods

2 - setter / getter for Filter
1 - getter DOMConfiguration
1 - getter for Async Mode
1 - getter for busy status
3 - parsing methods
1 - abort method

the exiting implementation does not support ASYNC mode of parsing. i teste

org.w3c.dom.ls

Interface LSParser

At 5th round get familier with DOMConfiguration object

`public interface LSParser`

An interface to an object that is able to build, or augment, a DOM tree from various input sources.

`LSParser` provides an API for parsing XML and building the corresponding DOM document structure. A `LSParser` instance can be obtained by invoking the `DOMImplementationLS.createLSParser()` method.

As specified in [DOM Level 3 Core] , when a document is first made available via the LSParser:

- there will never be two adjacent nodes of type NODE_TEXT, and there will never be empty text nodes.
- it is expected that the `value` and `nodeValue` attributes of an `Attr` node initially return the XML 1.0 normalized value. However, if the parameters "validate-if-schema" and "datatype-normalization" are set to `true`, depending on the attribute normalization used, the attribute values may differ from the ones obtained by the XML 1.0 attribute normalization. If the parameters "datatype-normalization" is set to `false`, the XML 1.0 attribute normalization is guaranteed to occur, and if the attributes list does not contain namespace declarations, the `attributes` attribute on `Element` node represents the property **[attributes]** defined in [XML Information Set] .

Asynchronous `LSParser` objects are expected to also implement the `events::EventTarget` interface so that event listeners can be registered on asynchronous `LSParser` objects.

Events supported by asynchronous `LSParser` objects are:

load

The `LSParser` finishes to load the document. See also the definition of the `LSLoadEvent` interface.

progress

The `LSParser` signals progress as data is parsed. This specification does not attempt to define exactly when progress events should be dispatched. That is intentionally left as implementation-dependent. Here is one example of how an application might dispatch progress events: Once the parser starts receiving data, a progress event is dispatched to indicate that the parsing starts. From there on, a progress event is dispatched for every 4096 bytes of data that is received and processed. This is only one example, though, and implementations can choose to dispatch progress events at any time while parsing, or not dispatch them at all. See also the definition of the `LSProgressEvent` interface.

Note: All events defined in this specification use the namespace URI `"http://www.w3.org/2002/DOM LS"`

While parsing an input source, errors are reported to the application through the error handler (`LSParser.domConfig`'s "error-handler" parameter). This specification does in no way try to define all possible errors that can occur while parsing XML, or any other markup, but some common error cases are defined. The types (`DOMError.type`) of errors and warnings defined by this specification are:

`"check-character-normalization-failure" [error]`

Raised if the parameter "check-character-normalization" is set to true and a string is encountered that fails normalization checking.

`"doctype-not-allowed" [fatal]`

Raised if the configuration parameter "disallow-doctype" is set to `true` and a doctype is encountered.

`"no-input-specified" [fatal]`

Raised when loading a document and no input is specified in the `LSInput` object.

`"pi-base-uri-not-preserved" [warning]`

Raised if a processing instruction is encountered in a location where the base URI of the processing instruction can not be preserved. One example of a case where this warning will be raised is if the configuration parameter "entities" is set to `false` and the following XML file is parsed:

```
<!DOCTYPE root [ <!ENTITY e SYSTEM 'subdir/myentity.ent' ]>
<root> &e; </root>
```

And `subdir/myentity.ent` contains:

```
<one> <two/> </one> <?pi 3.14159?>
 <more/>
```

"unbound-prefix-in-entity" [warning]

An implementation dependent warning that may be raised if the configuration parameter "namespaces" is set to `true` and an unbound namespace prefix is encountered in an entity's replacement text. Raising this warning is not enforced since some existing parsers may not recognize unbound namespace prefixes in the replacement text of entities.

"unknown-character-denormalization" [fatal]

Raised if the configuration parameter "ignore-unknown-character-denormalizations" is set to `false` and a character is encountered for which the processor cannot determine the normalization properties.

"unsupported-encoding" [fatal]

Raised if an unsupported encoding is encountered.

"unsupported-media-type" [fatal]

Raised if the configuration parameter "supported-media-types-only" is set to `true` and an unsupported media type is encountered.

In addition to raising the defined errors and warnings, implementations are expected to raise implementation specific errors and warnings for any other error and warning cases such as IO errors (file not found, permission denied,...), XML well-formedness errors, and so on.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Field Summary

static short	**ACTION_APPEND_AS_CHILDREN** Append the result of the parse operation as children of the context node.
static short	**ACTION_INSERT_AFTER** Insert the result of the parse operation as the immediately following sibling of the context node.
static short	**ACTION_INSERT_BEFORE** Insert the result of the parse operation as the immediately preceding sibling of the context node.
static short	**ACTION_REPLACE** Replace the context node with the result of the parse operation.
static short	**ACTION_REPLACE_CHILDREN** Replace all the children of the context node with the result of the parse operation.

Method Summary

void	**abort**() Abort the loading of the document that is currently being loaded by the LSParser
boolean	**getAsync**() true if the LSParser is asynchronous, false if it is synchronous.
boolean	**getBusy**() true if the LSParser is currently busy loading a document, otherwise false
DOMConfiguration	**getDomConfig**() The DOMConfiguration object used when parsing an input source.
LSParserFilter	**getFilter**() When a filter is provided, the implementation will call out to the filter as it is constructing the DOM tree structure.
Document	**parse**(LSInput input) Parse an XML document from a resource identified by a LSInput.
Document	**parseURI**(java.lang.String uri) Parse an XML document from a location identified by a URI reference [IETF RFC 2396].
Node	**parseWithContext**(LSInput input, Node contextArg, short action) Parse an XML fragment from a resource identified by a LSInput and insert the content into an existing document at the position specified with the context and action arguments.
void	**setFilter**(LSParserFilter filter) When a filter is provided, the implementation will call out to the filter as it is constructing the DOM tree structure.

Field Detail

ACTION_APPEND_AS_CHILDREN

this variable will be used parseWithContext()

`static final short` **`ACTION_APPEND_AS_CHILDREN`**

Append the result of the parse operation as children of the context node. For this action to work, the context node must be an `Element` or a `DocumentFragment`.

See Also:

Constant Field Values

---

this variable will be used parseWithContext()

ACTION_REPLACE_CHILDREN

`static final short` **`ACTION_REPLACE_CHILDREN`**

Replace all the children of the context node with the result of the parse operation. For this action to work, the context node must be an `Element`, a `Document`, or a `DocumentFragment`.

See Also:

Constant Field Values

---

this variable will be used parseWithContext()

ACTION_INSERT_BEFORE

`static final short` **`ACTION_INSERT_BEFORE`**

Insert the result of the parse operation as the immediately preceding sibling of the context node. For this action to work the context node's parent must be an `Element` or a `DocumentFragment`.

See Also:

Constant Field Values

---

this variable will be used parseWithContext()

ACTION_INSERT_AFTER

`static final short` **`ACTION_INSERT_AFTER`**

Insert the result of the parse operation as the immediately following sibling of the context node. For this action to work the context node's parent must be an `Element` or a `DocumentFragment`.

See Also:

Constant Field Values

ACTION_REPLACE

this variable will be used parseWithContext()

```
static final short ACTION_REPLACE
```

Replace the context node with the result of the parse operation. For this action to work, the context node must have a parent, and the parent must be an `Element` or a `DocumentFragment`.

See Also:

Constant Field Values

Method Detail

getDomConfig

```
DOMConfiguration getDomConfig()
```

The `DOMConfiguration` object used when parsing an input source. This `DOMConfiguration` is specific to the parse operation. No parameter values from this `DOMConfiguration` object are passed automatically to the `DOMConfiguration` object on the `Document` that is created, or used, by the parse operation. The DOM application is responsible for passing any needed parameter values from this `DOMConfiguration` object to the `DOMConfiguration` object referenced by the `Document` object.
In addition to the parameters recognized in on the DOMConfiguration interface defined in [DOM Level 3 Core] , the `DOMConfiguration` objects for `LSParser` add or modify the following parameters:

`"charset-overrides-xml-encoding"`
`true`
[*optional*] (*default*) If a higher level protocol such as HTTP [IETF RFC 2616]

provides an indication of the character encoding of the input stream being processed, that will override any encoding specified in the XML declaration or the Text declaration (see also section 4.3.3, "Character Encoding in Entities", in [XML 1.0]). Explicitly setting an encoding in the `LSInput` overrides any encoding from the protocol.

`false`

[*required*] The parser ignores any character set encoding information from higher-level protocols.

`"disallow-doctype"`

`true`

[*optional*] Throw a fatal **"doctype-not-allowed"** error if a doctype node is found while parsing the document. This is useful when dealing with things like SOAP envelopes where doctype nodes are not allowed.

`false`

[*required*] (*default*) Allow doctype nodes in the document.

`"ignore-unknown-character-denormalizations"`

`true`

[*required*] (*default*) If, while verifying full normalization when [XML 1.1] is supported, a processor encounters characters for which it cannot determine the normalization properties, then the processor will ignore any possible denormalizations caused by these characters. This parameter is ignored for [XML 1.0].

`false`

[*optional*] Report an fatal **"unknown-character-denormalization"** error if a character is encountered for which the processor cannot determine the normalization properties.

`"infoset"`

See the definition of `DOMConfiguration` for a description of this parameter. Unlike in [DOM Level 3 Core] , this parameter will default to `true` for `LSParser`.

`"namespaces"`

`true`

[*required*] (*default*) Perform the namespace processing as defined in [XML Namespaces] and [XML Namespaces 1.1] .

`false`

[*optional*] Do not perform the namespace processing.

`"resource-resolver"`

[*required*] A reference to a `LSResourceResolver` object, or null. If the value of this parameter is not null when an external resource (such as an external XML entity or an XML schema location) is encountered, the implementation will request that the `LSResourceResolver` referenced in this parameter resolves the resource.

`"supported-media-types-only"`

`true`

[*optional*] Check that the media type of the parsed resource is a supported media type. If an unsupported media type is encountered, a fatal error of type "**unsupported-media-type**" will be raised. The media types defined in [IETF RFC 3023] must always be accepted.

`false`

[*required*] (*default*) Accept any media type.

`"validate"`

See the definition of `DOMConfiguration` for a description of this parameter. Unlike in [DOM Level 3 Core] , the processing of the internal subset is always accomplished, even if this parameter is set to `false`.

`"validate-if-schema"`

See the definition of `DOMConfiguration` for a description of this parameter. Unlike in [DOM Level 3 Core] , the processing of the internal subset is always accomplished, even if this parameter is set to `false`.

`"well-formed"`

See the definition of `DOMConfiguration` for a description of this parameter. Unlike in [DOM Level 3 Core] , this parameter cannot be set to `false`.

---

getFilter

`LSParserFilter` **`getFilter()`**

When a filter is provided, the implementation will call out to the filter as it is constructing the DOM tree structure. The filter can choose to remove elements from the document being constructed, or to terminate the parsing early.

The filter is invoked after the operations requested by the `DOMConfiguration` parameters have been applied. For example, if "validate" is set to `true`, the validation is done before invoking the filter.

---

setFilter

`void` **`setFilter`**`(LSParserFilter filter)`

~~When a filter is provided, the implementation will call out to the filter as it is constructing the DOM tree structure. The filter can choose to remove elements from the document being constructed, or to terminate the parsing early.~~

The filter is invoked after the operations requested by the `DOMConfiguration` parameters have been applied. For example, if "validate" is set to `true`, the validation is done before invoking the filter.

---

getAsync

```
boolean getAsync()
```

`true` if the `LSParser` is asynchronous, `false` if it is synchronous.

---

getBusy

```
boolean getBusy()
```

`true` if the `LSParser` is currently busy loading a document, otherwise `false`.

---

parse

```
Document parse(LSInput input)
               throws DOMException,
                      LSException
```

Parse an XML document from a resource identified by a `LSInput`.

Parameters:

`input` - The `LSInput` from which the source of the document is to be read.

Returns:

If the `LSParser` is a synchronous `LSParser`, the newly created and populated `Document` is returned. If the `LSParser` is asynchronous, `null` is returned since the document object may not yet be constructed when this method returns.

Throws:

`DOMException` - INVALID_STATE_ERR: Raised if the `LSParser`'s `LSParser.busy` attribute is `true`.

LSException - PARSE_ERR: Raised if the LSParser was unable to load the XML document. DOM applications should attach a DOMErrorHandler using the parameter "error-handler" if they wish to get details on the error.

---

parseURI

```
Document parseURI(java.lang.String uri)
                  throws DOMException,
                         LSException
```

Parse an XML document from a location identified by a URI reference [IETF RFC 2396]. If the URI contains a fragment identifier (see section 4.1 in [IETF RFC 2396]), the behavior is not defined by this specification, future versions of this specification may define the behavior.

Parameters:

uri - The location of the XML document to be read.

Returns:

If the LSParser is a synchronous LSParser, the newly created and populated Document is returned, or null if an error occured. If the LSParser is asynchronous, null is returned since the document object may not yet be constructed when this method returns.

Throws:

DOMException - INVALID_STATE_ERR: Raised if the LSParser.busy attribute is true

LSException - PARSE_ERR: Raised if the LSParser was unable to load the XML document. DOM applications should attach a DOMErrorHandler using the parameter "error-handler" if they wish to get details on the error.

---

parseWithContext

i dont want use this method.

```
Node parseWithContext(LSInput input,
                      Node contextArg,
                      short action)
                      throws DOMException,
                             LSException
```

Parse an XML fragment from a resource identified by a `LSInput` and insert the content into an existing document at the position specified with the `context` and `action` arguments. When parsing the input stream, the context node (or its parent, depending on where the result will be inserted) is used for resolving unbound namespace prefixes. The context node's `ownerDocument` node (or the node itself if the node of type `DOCUMENT_NODE`) is used to resolve default attributes and entity references.
As the new data is inserted into the document, at least one mutation event is fired per new immediate child or sibling of the context node.
If the context node is a `Document` node and the action is `ACTION_REPLACE_CHILDREN`, then the document that is passed as the context node will be changed such that its `xmlEncoding`, `documentURI`, `xmlVersion`, `inputEncoding`, `xmlStandalone`, and all other such attributes are set to what they would be set to if the input source was parsed using `LSParser.parse()`.
This method is always synchronous, even if the `LSParser` is asynchronous (`LSParser.async` is `true`).
If an error occurs while parsing, the caller is notified through the `ErrorHandler` instance associated with the "error-handler" parameter of the `DOMConfiguration`.
When calling `parseWithContext`, the values of the following configuration parameters will be ignored and their default values will always be used instead: "validate", "validate-if-schema", and "element-content-whitespace". Other parameters will be treated normally, and the parser is expected to call the `LSParserFilter` just as if a whole document was parsed.

Parameters:

`input` - The `LSInput` from which the source document is to be read. The source document must be an XML fragment, i.e. anything except a complete XML document (except in the case where the context node of type `DOCUMENT_NODE`, and the action is `ACTION_REPLACE_CHILDREN`), a DOCTYPE (internal subset), entity declaration(s), notation declaration(s), or XML or text declaration(s).

`contextArg` - The node that is used as the context for the data that is being parsed. This node must be a `Document` node, a `DocumentFragment` node, or a node of a type that is allowed as a child of an `Element` node, e.g. it cannot be an `Attribute` node.

`action` - This parameter describes which action should be taken between the new set of nodes being inserted and the existing children of the context node. The set of possible actions is defined in `ACTION_TYPES` above.

Returns:

Return the node that is the result of the parse operation. If the result is more than one top-level node, the first one is returned.

Throws:

`DOMException` - HIERARCHY_REQUEST_ERR: Raised if the content cannot

replace, be inserted before, after, or as a child of the context node (see also `Node.insertBefore` or `Node.replaceChild` in [DOM Level 3 Core]).
NOT_SUPPORTED_ERR: Raised if the `LSParser` doesn't support this method, or if the context node is of type `Document` and the DOM implementation doesn't support the replacement of the `DocumentType` child or `Element` child.
NO_MODIFICATION_ALLOWED_ERR: Raised if the context node is a read only node and the content is being appended to its child list, or if the parent node of the context node is read only node and the content is being inserted in its child list.
INVALID_STATE_ERR: Raised if the `LSParser.busy` attribute is `true`.
`LSException` - PARSE_ERR: Raised if the `LSParser` was unable to load the XML fragment. DOM applications should attach a `DOMErrorHandler` using the parameter "error-handler" if they wish to get details on the error.

---

abort

`void abort()`

Abort the loading of the document that is currently being loaded by the `LSParser`. If the `LSParser` is currently not busy, a call to this method does nothing.

---

24 - Oct - 08

it has only 3 methods

24 - Dec - 08

org.w3c.dom.ls

Interface LSParserFilter

WhatToShow is rep. what type of node has to apply for filter. remaining type of node processed automatically without appling filter

```
public interface LSParserFilter
```

~~`LSParserFilters` provide applications the ability to examine nodes as they are being constructed while parsing. As each node is examined, it may be modified or removed, or the entire parse may be terminated early.~~

At the time any of the filter methods are called by the parser, the owner Document and DOMImplementation objects exist and are accessible. The document element is never passed to the `LSParserFilter` methods, i.e. it is not possible to filter out the document element. `Document`, `DocumentType`, `Notation`, `Entity`, and `Attr` nodes are never passed to the `acceptNode` method on the filter. The child nodes of an `EntityReference` node are passed to the filter if the parameter "entities" is set to `false`. Note that, as described by the parameter "entities", unexpanded entity reference nodes are never discarded and are always passed to the filter.

~~All validity checking while parsing a document occurs on the source document as it appears on the input stream, not on the DOM document as it is built in memory. With filters, the document in memory may be a subset of the document on the stream, and its validity may have been affected by the filtering.~~

All default attributes must be present on elements when the elements are passed to the filter methods. All other default content must be passed to the filter methods.

DOM applications must not raise exceptions in a filter. The effect of throwing exceptions from a filter is DOM implementation dependent.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Field Summary

static short	**FILTER_ACCEPT** Accept the node.
static short	**FILTER_INTERRUPT** Interrupt the normal processing of the document.
static short	**FILTER_REJECT** Reject the node and its children.
static short	**FILTER_SKIP** Skip this single node.

Method Summary

short	**acceptNode**(Node nodeArg) This method will be called by the parser at the completion of the parsing of each node.
int	**getWhatToShow**() Tells the LSParser what types of nodes to show to the method LSParserFilter.acceptNode.
short	**startElement**(Element elementArg) The parser will call this method after each Element start tag has been scanned, but before the remainder of the Element is processed.

Field Detail

FILTER_ACCEPT

```
static final short FILTER_ACCEPT
```

Accept the node.

See Also:

Constant Field Values

FILTER_REJECT

`static final short` **`FILTER_REJECT`**

Reject the node and its children.

See Also:
Constant Field Values

FILTER_SKIP

`static final short` **`FILTER_SKIP`**

Skip this single node. The children of this node will still be considered.

See Also:
Constant Field Values

FILTER_INTERRUPT

`static final short` **`FILTER_INTERRUPT`**

Interrupt the normal processing of the document.

See Also:
Constant Field Values

Method Detail

startElement

`short` **`startElement`**`(Element elementArg)`

The parser will call this method after each `Element` start tag has been scanned, but before the remainder of the `Element` is processed. The intent is to allow the element, including any children, to be efficiently skipped. Note that only element nodes are passed to the

`startElement` function.
The element node passed to `startElement` for filtering will include all of the Element's attributes, but none of the children nodes. The Element may not yet be in place in the document being constructed (it may not have a parent node.)
A `startElement` filter function may access or change the attributes for the Element.
Changing Namespace declarations will have no effect on namespace resolution by the parser.
For efficiency, the Element node passed to the filter may not be the same one as is actually placed in the tree if the node is accepted. And the actual node (node object identity) may be reused during the process of reading in and filtering a document.

Parameters:

`elementArg` - The newly encountered element. At the time this method is called, the element is incomplete - it will have its attributes, but no children.

Returns:

- `FILTER_ACCEPT` if the `Element` should be included in the DOM document being built.
- `FILTER_REJECT` if the `Element` and all of its children should be rejected.
- `FILTER_SKIP` if the `Element` should be skipped. All of its children are inserted in place of the skipped `Element` node.
- `FILTER_INTERRUPT` if the filter wants to stop the processing of the document. Interrupting the processing of the document does no longer guarantee that the resulting DOM tree is XML well-formed. The `Element` is rejected.

Returning any other values will result in unspecified behavior.

---

acceptNode

`short` **`acceptNode`**`(`Node `nodeArg)`

This method will be called by the parser at the completion of the parsing of each node. The node and all of its descendants will exist and be complete. The parent node will also exist, although it may be incomplete, i.e. it may have additional children that have not yet been parsed. Attribute nodes are never passed to this function.
From within this method, the new node may be freely modified - children may be added or removed, text nodes modified, etc. The state of the rest of the document outside this node is not defined, and the affect of any attempt to navigate to, or to modify any other part of the document is undefined.
For validating parsers, the checks are made on the original document, before any modification by the filter. No validity checks are made on any document modifications made by the filter.
If this new node is rejected, the parser might reuse the new node and any of its descendants.

Parameters:

`nodeArg` - The newly constructed element. At the time this method is called, the element is complete - it has all of its children (and their children, recursively) and attributes, and is attached as a child to its parent.

Returns:

- `FILTER_ACCEPT` if this `Node` should be included in the DOM document being built.
- `FILTER_REJECT` if the `Node` and all of its children should be rejected.
- `FILTER_SKIP` if the `Node` should be skipped and the `Node` should be replaced by all the children of the `Node`.
- `FILTER_INTERRUPT` if the filter wants to stop the processing of the document. Interrupting the processing of the document does no longer guarantee that the resulting DOM tree is XML well-formed. The `Node` is accepted and will be the last completely parsed node.

---

getWhatToShow

`int` **`getWhatToShow`**`()`

Tells the `LSParser` what types of nodes to show to the method `LSParserFilter.acceptNode`. If a node is not shown to the filter using this attribute, it is automatically included in the DOM document being built. See `NodeFilter` for definition of the constants. The constants `SHOW_ATTRIBUTE` , `SHOW_DOCUMENT`, `SHOW_DOCUMENT_TYPE`, `SHOW_NOTATION`, `SHOW_ENTITY`, and `SHOW_DOCUMENT_FRAGMENT` are meaningless here. Those nodes will never be passed to `LSParserFilter.acceptNode`.
The constants used here are defined in [DOM Level 2 Traversal and Range] .

---

24 - Oct - 08

org.w3c.dom.ls

Interface LSProgressEvent

it has only 3 methods. all are getter methods only

All Superinterfaces:

Event

this event can be evaluated only if parser is in ASYNCHRONOUS mode.

unfortunatly, the existing JAXP support only SYNCHRONOUS mode

IN 5th Round at least get to know how to run in ASync mode

```
public interface LSProgressEvent
```

extends Event

This interface represents a progress event object that notifies the application about progress as a document is parsed. It extends the `Event` interface defined in [DOM Level 3 Events] .

The units used for the attributes `position` and `totalSize` are not specified and can be implementation and input dependent.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Field Summary

Fields inherited from interface org.w3c.dom.events.Event
AT_TARGET, BUBBLING_PHASE, CAPTURING_PHASE

Method Summary

LSInput	**getInput**() The input source that is being parsed.

int	**getPosition**() The current position in the input source, including all external entities and other resources that have been read.
int	**getTotalSize**() The total size of the document including all external resources, this number might change as a document is being parsed if references to more external resources are seen.

Methods inherited from interface org.w3c.dom.events.Event
getBubbles, getCancelable, getCurrentTarget, getEventPhase, getTarget, getTimeStamp, getType, initEvent, preventDefault, stopPropagation

Method Detail

getInput

LSInput **getInput**()

The input source that is being parsed.

---

getPosition

int **getPosition**()

The current position in the input source, including all external entities and other resources that have been read.

---

getTotalSize

int **getTotalSize**()

The total size of the document including all external resources, this number might change as a

document is being parsed if references to more external resources are seen. A value of `0` is returned if the total size cannot be determined or estimated.

---

24 - Oct - 08

it is not need.
since it is DTD related.

Mostly all of the resolver in any of JAXP is DTD related. just skip at 5th round study

org.w3c.dom.ls

Interface LSR

this is used by Validation API.. study at 5th round

```
public interface LSResourceResolver
```

`LSResourceResolver` provides a way for applications to redirect references to external resources.

Applications needing to implement custom handling for external resources can implement this interface and register their implementation by setting the "resource-resolver" parameter of `DOMConfiguration` objects attached to `LSParser` and `LSSerializer`. It can also be register on `DOMConfiguration` objects attached to `Document` if the "LS" feature is supported.

The `LSParser` will then allow the application to intercept any external entities, including the external DTD subset and external parameter entities, before including them. The top-level document entity is never passed to the `resolveResource` method.

Many DOM applications will not need to implement this interface, but it will be especially useful for applications that build XML documents from databases or other specialized input sources, or for applications that use URNs.

Note: `LSResourceResolver` is based on the SAX2 [SAX] `EntityResolver` interface.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Method Summary

LSInput	**resolveResource**(java.lang.String type, java.lang.String namespaceURI, java.lang.String publicId, java.lang.String systemId, java.lang.String baseURI) Allow the application to resolve external resources.

Method Detail

resolveResource

```
LSInput resolveResource(java.lang.String type,
                        java.lang.String namespaceURI,
                        java.lang.String publicId,
                        java.lang.String systemId,
                        java.lang.String baseURI)
```

Allow the application to resolve external resources.
The `LSParser` will call this method before opening any external resource, including the external DTD subset, external entities referenced within the DTD, and external entities referenced within the document element (however, the top-level document entity is not passed to this method). The application may then request that the `LSParser` resolve the external resource itself, that it use an alternative URI, or that it use an entirely different input source.
Application writers can use this method to redirect external system identifiers to secure and/or local URI, to look up public identifiers in a catalogue, or to read an entity from a database or other input source (including, for example, a dialog box).

Parameters:

`type` - The type of the resource being resolved. For XML [XML 1.0] resources (i.e. entities), applications must use the value `"http://www.w3.org/TR/REC-xml"`. For XML Schema [XML Schema Part 1] , applications must use the value `"http://www.w3.org/2001/XMLSchema"`. Other types of resources are outside the scope of this specification and therefore should recommend an absolute URI in order to use this method.

`namespaceURI` - The namespace of the resource being resolved, e.g. the target namespace of the XML Schema [XML Schema Part 1] when resolving XML Schema resources.

`publicId` - The public identifier of the external entity being referenced, or `null` if no public identifier was supplied or if the resource is not an entity.

`systemId` - The system identifier, a URI reference [IETF RFC 2396], of the external resource being referenced, or `null` if no system identifier was supplied.

`baseURI` - The absolute base URI of the resource being parsed, or `null` if there is no base URI.

Returns:

A `LSInput` object describing the new input source, or `null` to request that the parser open a regular URI connection to the resource.

At 5th round get familier with DOMConfiguration object

it has 8 methods

2 - getter / setter for Filter
2 - getter / setter for newline
1 - getter for DOM config
3 - overloaded Write method

org.w3c.dom.ls

Interface LSSerializer

`public interface LSSerializer`

A `LSSerializer` provides an API for serializing (writing) a DOM document out into XML. The XML data is written to a string or an output stream. Any changes or fixups made during the serialization affect only the serialized data. The `Document` object and its children are never altered by the serialization operation.

During serialization of XML data, namespace fixup is done as defined in [DOM Level 3 Core] , Appendix B. [DOM Level 2 Core] allows empty strings as a real namespace URI. If the `namespaceURI` of a `Node` is empty string, the serialization will treat them as `null`, ignoring the prefix if any.

`LSSerializer` accepts any node type for serialization. For nodes of type `Document` or `Entity`, well-formed XML will be created when possible (well-formedness is guaranteed if the document or entity comes from a parse operation and is unchanged since it was created). The serialized output for these node types is either as a XML document or an External XML Entity, respectively, and is acceptable input for an XML parser. For all other types of nodes the serialized form is implementation dependent.

Within a `Document`, `DocumentFragment`, or `Entity` being serialized, `Nodes` are processed as follows

- `Document` nodes are written, including the XML declaration (unless the parameter "xml-declaration" is set to `false`) and a DTD subset, if one exists in the DOM. Writing a `Document` node serializes the entire document.
- `Entity` nodes, when written directly by `LSSerializer.write`, outputs the entity expansion but no namespace fixup is done. The resulting output will be valid as an external entity.
- If the parameter "entities" is set to `true`, `EntityReference` nodes are serialized as an entity

reference of the form "`&entityName;`" in the output. Child nodes (the expansion) of the entity reference are ignored. If the parameter "entities" is set to `false`, only the children of the entity reference are serialized. `EntityReference` nodes with no children (no corresponding `Entity` node or the corresponding `Entity` nodes have no children) are always serialized.

- `CDATASections` containing content characters that cannot be represented in the specified output encoding are handled according to the "split-cdata-sections" parameter. If the parameter is set to `true`, `CDATASections` are split, and the unrepresentable characters are serialized as numeric character references in ordinary content. The exact position and number of splits is not specified. If the parameter is set to `false`, unrepresentable characters in a `CDATASection` are reported as `"wf-invalid-character"` errors if the parameter "well-formed" is set to `true`. The error is not recoverable - there is no mechanism for supplying alternative characters and continuing with the serialization.
- `DocumentFragment` nodes are serialized by serializing the children of the document fragment in the order they appear in the document fragment.
- All other node types (Element, Text, etc.) are serialized to their corresponding XML source form.

Note: The serialization of a `Node` does not always generate a well-formed XML document, i.e. a `LSParser` might throw fatal errors when parsing the resulting serialization.

Within the character data of a document (outside of markup), any characters that cannot be represented directly are replaced with character references. Occurrences of '<' and '&' are replaced by the predefined entities < and &. The other predefined entities (>, ', and ") might not be used, except where needed (e.g. using > in cases such as ']]>'). Any characters that cannot be represented directly in the output character encoding are serialized as numeric character references (and since character encoding standards commonly use hexadecimal representations of characters, using the hexadecimal representation when serializing character references is encouraged).

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "'", and the double-quote character (") as """. New line characters and other characters that cannot be represented directly in attribute values in the output character encoding are serialized as a numeric character reference.

Within markup, but outside of attributes, any occurrence of a character that cannot be represented in the output character encoding is reported as a `DOMError` fatal error. An example would be serializing the element <LaCañada/> with `encoding="us-ascii"`. This will result with a generation of a `DOMError` "wf-invalid-character-in-node-name" (as proposed in "well-formed").

When requested by setting the parameter "normalize-characters" on `LSSerializer` to true, character normalization is performed according to the definition of fully normalized characters included in appendix E of [XML 1.1] on all data to be serialized, both markup and character data. The character

normalization process affects only the data as it is being written; it does not alter the DOM's view of the document after serialization has completed.

Implementations are required to support the encodings "UTF-8", "UTF-16", "UTF-16BE", and "UTF-16LE" to guarantee that data is serializable in all encodings that are required to be supported by all XML parsers. When the encoding is UTF-8, whether or not a byte order mark is serialized, or if the output is big-endian or little-endian, is implementation dependent. When the encoding is UTF-16, whether or not the output is big-endian or little-endian is implementation dependent, but a Byte Order Mark must be generated for non-character outputs, such as `LSOutput.byteStream` or `LSOutput.systemId`. If the Byte Order Mark is not generated, a "byte-order-mark-needed" warning is reported. When the encoding is UTF-16LE or UTF-16BE, the output is big-endian (UTF-16BE) or little-endian (UTF-16LE) and the Byte Order Mark is not be generated. In all cases, the encoding declaration, if generated, will correspond to the encoding used during the serialization (e.g. `encoding="UTF-16"` will appear if UTF-16 was requested).

Namespaces are fixed up during serialization, the serialization process will verify that namespace declarations, namespace prefixes and the namespace URI associated with elements and attributes are consistent. If inconsistencies are found, the serialized form of the document will be altered to remove them. The method used for doing the namespace fixup while serializing a document is the algorithm defined in Appendix B.1, "Namespace normalization", of [DOM Level 3 Core] .

While serializing a document, the parameter "discard-default-content" controls whether or not non-specified data is serialized.

While serializing, errors and warnings are reported to the application through the error handler (`LSSerializer.domConfig`'s "error-handler" parameter). This specification does in no way try to define all possible errors and warnings that can occur while serializing a DOM node, but some common error and warning cases are defined. The types (`DOMError.type`) of errors and warnings defined by this specification are:

`"no-output-specified" [fatal]`
: Raised when writing to a `LSOutput` if no output is specified in the `LSOutput`

`"unbound-prefix-in-entity-reference" [fatal]`
: Raised if the configuration parameter "namespaces" is set to `true` and an entity whose replacement text contains unbound namespace prefixes is referenced in a location where there are no bindings for the namespace prefixes.

`"unsupported-encoding" [fatal]`
: Raised if an unsupported encoding is encountered.

In addition to raising the defined errors and warnings, implementations are expected to raise implementation specific errors and warnings for any other error and warning cases such as IO errors (file

not found, permission denied,...) and so on.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Method Summary

DOMConfiguration	**getDomConfig**() The DOMConfiguration object used by the LSSerializer when serializing a DOM node.
LSSerializerFilter	**getFilter**() When the application provides a filter, the serializer will call out to the filter before serializing each Node.
java.lang.String	**getNewLine**() The end-of-line sequence of characters to be used in the XML being written out.
void	**setFilter**(LSSerializerFilter filter) When the application provides a filter, the serializer will call out to the filter before serializing each Node.
void	**setNewLine**(java.lang.String newLine) The end-of-line sequence of characters to be used in the XML being written out.
boolean	**write**(Node nodeArg, LSOutput destination) Serialize the specified node as described above in the general description of the LSSerializer interface.
java.lang.String	**writeToString**(Node nodeArg) Serialize the specified node as described above in the general description of the LSSerializer interface.
boolean	**writeToURI**(Node nodeArg, java.lang.String uri) A convenience method that acts as if LSSerializer.write was called with a LSOutput with no encoding specified and LSOutput.systemId set to the uri argument.

Method Detail

getDomConfig

`DOMConfiguration` **`getDomConfig()`**

The `DOMConfiguration` object used by the `LSSerializer` when serializing a DOM node. In addition to the parameters recognized by the DOMConfiguration interface defined in [DOM Level 3 Core] , the `DOMConfiguration` objects for `LSSerializer` adds, or modifies, the following parameters:

`"canonical-form"`

`true`

[*optional*] Writes the document according to the rules specified in [Canonical XML]. In addition to the behavior described in " canonical-form" [DOM Level 3 Core] , setting this parameter to `true` will set the parameters "format-pretty-print", "discard-default-content", and "xml-declaration ", to `false`. Setting one of those parameters to `true` will set this parameter to `false`. Serializing an XML 1.1 document when "canonical-form" is `true` will generate a fatal error.

`false`

[*required*] (*default*) Do not canonicalize the output.

`"discard-default-content"`

`true`

[*required*] (*default*) Use the `Attr.specified` attribute to decide what attributes should be discarded. Note that some implementations might use whatever information available to the implementation (i.e. XML schema, DTD, the `Attr.specified` attribute, and so on) to determine what attributes and content to discard if this parameter is set to `true`.

`false`

[*required*]Keep all attributes and all content.

`"format-pretty-print"`

`true`

[*optional*] Formatting the output by adding whitespace to produce a pretty-printed, indented, human-readable form. The exact form of the transformations is not specified by this specification. Pretty-printing changes the content of the document and may affect the validity of the document, validating implementations should preserve validity.

`false`

[*required*] (*default*) Don't pretty-print the result.

`"ignore-unknown-character-denormalizations"`

`true`

[*required*] (*default*) If, while verifying full normalization when [XML 1.1] is supported, a character is encountered for which the normalization properties cannot be determined, then raise a `"unknown-character-denormalization"` warning (instead of raising an error, if this parameter is not set) and ignore any possible denormalizations caused by these characters.

`false`

[*optional*] Report a fatal error if a character is encountered for which the processor cannot determine the normalization properties.

`"normalize-characters"`

This parameter is equivalent to the one defined by `DOMConfiguration` in [DOM Level 3 Core] . Unlike in the Core, the default value for this parameter is `true`. While DOM implementations are not required to support fully normalizing the characters in the document according to appendix E of [XML 1.1], this parameter must be activated by default if supported.

`"xml-declaration"`

`true`

[*required*] (*default*) If a `Document`, `Element`, or `Entity` node is serialized, the XML declaration, or text declaration, should be included. The version (`Document.xmlVersion` if the document is a Level 3 document and the version is non-null, otherwise use the value "1.0"), and the output encoding (see `LSSerializer.write` for details on how to find the output encoding) are specified in the serialized XML declaration.

`false`

[*required*] Do not serialize the XML and text declarations. Report a `"xml-declaration-needed"` warning if this will cause problems (i.e. the serialized data is of an XML version other than [XML 1.0], or an encoding would be needed to be able to re-parse the serialized data).

---

getNewLine

`java.lang.String` **`getNewLine`**`()`

~~The end-of-line sequence of characters to be used in the XML being written out. Any string is supported, but XML treats only a certain set of characters sequence as end-of-line (See section 2.11, "End-of-Line Handling" in [XML 1.0], if the serialized content is XML 1.0 or section 2.11, "End-of-Line Handling" in [XML 1.1], if the serialized content is XML 1.1). Using other character sequences than the recommended ones can result in a document that is either not serializable or not well-formed).~~

~~On retrieval, the default value of this attribute is the implementation specific default end-of-line~~

sequence. DOM implementations should choose the default to match the usual convention for text files in the environment being used. Implementations must choose a default sequence that matches one of those allowed by XML 1.0 or XML 1.1, depending on the serialized content. Setting this attribute to `null` will reset its value to the default value.

---

setNewLine

`void` **`setNewLine`**`(java.lang.String newLine)`

The end-of-line sequen it can be set any arbitrary value ML being written out. Any string is supported, but XML treats only a certain set of characters sequence as end-of-line (See section 2.11, "End-of-Line Handling" in [XML 1.0], if the serialized content is XML 1.0 or section 2.11, "End-of-Line Handling" in [XML 1.1], if the serialized content is XML 1.1). Using other character sequences than the recommended ones can result in a document that is either not serializable or not well-formed).
On retrieval, the default value of this attribute is the implementation specific default end-of-line sequence. DOM implementations should choose the default to match the usual convention for text files in the environment being used. Implementations must choose a default sequence that matches one of those allowed by XML 1.0 or XML 1.1, depending on the serialized content. Setting this attribute to `null` will reset its value to the default value.

---

getFilter

`LSSerializerFilter` **`getFilter`**`()`

When the application provides a filter, the serializer will call out to the filter before serializing each Node. The filter implementation can choose to remove the node from the stream or to terminate the serialization early.
The filter is invoked after the operations requested by the `DOMConfiguration` parameters have been applied. For example, CDATA sections won't be passed to the filter if " cdata-sections" is set to `false`.

---

setFilter

```
void setFilter(LSSerializerFilter filter)
```

When the application provides a filter, the serializer will call out to the filter before serializing each Node. The filter implementation can choose to remove the node from the stream or to terminate the serialization early.
The filter is invoked after the operations requested by the `DOMConfiguration` parameters have been applied. For example, CDATA sections won't be passed to the filter if "cdata-sections" is set to `false`.

---

write

```
boolean write(Node nodeArg,
              LSOutput destination)
              throws LSException
```

Serialize the specified node as described above in the general description of the `LSSerializer` interface. The output is written to the supplied `LSOutput`.
When writing to a `LSOutput`, the encoding is found by looking at the encoding information that is reachable through the `LSOutput` and the item to be written (or its owner document) in this order:

1. `LSOutput.encoding`,
2. `Document.inputEncoding`,
3. `Document.xmlEncoding`.

If no encoding is reachable through the above properties, a default encoding of "UTF-8" will be used. If the specified encoding is not supported an "unsupported-encoding" fatal error is raised.
If no output is specified in the `LSOutput`, a "no-output-specified" fatal error is raised.
The implementation is responsible of associating the appropriate media type with the serialized data.
When writing to a HTTP URI, a HTTP PUT is performed. When writing to other types of URIs, the mechanism for writing the data to the URI is implementation dependent.

Parameters:

`nodeArg` - The node to serialize.

`destination` - The destination for the serialized DOM.

Returns:

Returns `true` if `node` was successfully serialized. Return `false` in case the normal processing stopped but the implementation kept serializing the document; the result of the

serialization being implementation dependent then.

Throws:

`LSException` - SERIALIZE_ERR: Raised if the `LSSerializer` was unable to serialize the node. DOM applications should attach a `DOMErrorHandler` using the parameter " error-handler" if they wish to get details on the error.

---

writeToURI

```
boolean writeToURI(Node nodeArg,
                   java.lang.String uri)
                   throws LSException
```

A convenience method that acts as if `LSSerializer.write` was called with a `LSOutput` with no encoding specified and `LSOutput.systemId` set to the `uri` argument.

Parameters:

`nodeArg` - The node to serialize.

`uri` - The URI to write to.

Returns:

Returns `true` if `node` was successfully serialized. Return `false` in case the normal processing stopped but the implementation kept serializing the document; the result of the serialization being implementation dependent then.

Throws:

`LSException` - SERIALIZE_ERR: Raised if the `LSSerializer` was unable to serialize the node. DOM applications should attach a `DOMErrorHandler` using the parameter " error-handler" if they wish to get details on the error.

---

writeToString

```
java.lang.String writeToString(Node nodeArg)
                               throws DOMException,
                                      LSException
```

Serialize the specified node as described above in the general description of the `LSSerializer` interface. The output is written to a `DOMString` that is returned to the caller.

The encoding used is the encoding of the `DOMString` type, i.e. UTF-16. Note that no Byte Order Mark is generated in a `DOMString` object.

Parameters:

`nodeArg` - The node to serialize.

Returns:

Returns the serialized data.

Throws:

`DOMException` - DOMSTRING_SIZE_ERR: Raised if the resulting string is too long to fit in a `DOMString`.

`LSException` - SERIALIZE_ERR: Raised if the `LSSerializer` was unable to serialize the node. DOM applications should attach a `DOMErrorHandler` using the parameter " error-handler" if they wish to get details on the error.

---

25 - Dec - 08

org.w3c.dom.ls

Interface LSSerializerFilter

it has only 2 method

this page itself missed acceptNode method

All Superinterfaces:

NodeFilter

WhatToShow is rep. what type of node has to apply for filter. remaining type of node processed automatically without appling filter

```
public interface LSSeria
```

extends NodeFilter

`LSSerializerFilter`s provide applications the ability to examine nodes as they are being serialized and decide what nodes should be serialized or not. The `LSSerializerFilter` interface is based on the `NodeFilter` interface defined in [DOM Level 2 Traversal and Range] .

`Document`, `DocumentType`, `DocumentFragment`, `Notation`, `Entity` , and children of `Attr` nodes are not passed to the filter. The child nodes of an `EntityReference` node are only passed to the filter if the `EntityReference` node is skipped by the method `LSParserFilter.acceptNode()`.

When serializing an `Element`, the element is passed to the filter before any of its attributes are passed to the filter. Namespace declaration attributes, and default attributes (except in the case when " discard-default-content" is set to `false`), are never passed to the filter.

The result of any attempt to modify a node passed to a `LSSerializerFilter` is implementation dependent.

DOM applications must not raise exceptions in a filter. The effect of throwing exceptions from a filter is DOM implementation dependent.

For efficiency, a node passed to the filter may not be the same as the one that is actually in the tree. And the actual node (node object identity) may be reused during the process of filtering and serializing a document.

See also the Document Object Model (DOM) Level 3 Load and Save Specification.

Field Summary

Fields inherited from interface org.w3c.dom.traversal.NodeFilter
FILTER_ACCEPT, FILTER_REJECT, FILTER_SKIP, SHOW_ALL, SHOW_ATTRIBUTE, SHOW_CDATA_SECTION, SHOW_COMMENT, SHOW_DOCUMENT, SHOW_DOCUMENT_FRAGMENT, SHOW_DOCUMENT_TYPE, SHOW_ELEMENT, SHOW_ENTITY, SHOW_ENTITY_REFERENCE, SHOW_NOTATION, SHOW_PROCESSING_INSTRUCTION, SHOW_TEXT

Method Summary

int	**getWhatToShow**() Tells the LSSerializer what types of nodes to show to the filter.

Methods inherited from interface org.w3c.dom.traversal.NodeFilter
acceptNode

Method Detail

getWhatToShow

`int getWhatToShow()`

so, at a time it can use only one type of node

Tells the LSSerializer what types of nodes to show to the filter. If a node is not shown to the filter using this attribute, it is automatically serialized. See NodeFilter for definition of the constants. The constants SHOW_DOCUMENT , SHOW_DOCUMENT_TYPE, SHOW_DOCUMENT_FRAGMENT , SHOW_NOTATION, and SHOW_ENTITY are meaningless here, such nodes will never be passed to a LSSerializerFilter.

yes

Unlike [DOM Level 2 Traversal and Range] , the SHOW_ATTRIBUTE constant indicates that the

`Attr` nodes are shown and passed to the filter.

The constants used here are defined in [DOM Level 2 Traversal and Range] .

---

it has 8 methods

There are 6 methods, based Name as parameter
2 - set (add)
2 - get
2 - remove

so the DOM has two collection for holding Node

1. NodeList
2. NamedNodeMap

This is COLLECTION based map for holding Node

the interface name itself says, it can associate any sub-type of node with name as key

org.w3c.dom

Interface NamedNodeMap

mostly we use this interface only access of attribute nodes.
Since the any node if implement this interface, it can be collected in NamedNodeMap.
So, even if we use for attribute, it is not only for that.

`public interface NamedNodeMap`

Objects implementing the `NamedNodeMap` interface are used to represent collections of nodes that can be accessed by name. Note that `NamedNodeMap` does not inherit from `NodeList`; `NamedNodeMaps` are not maintained in any particular order. Objects contained in an object implementing `NamedNodeMap` may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a `NamedNodeMap`, and does not imply that the DOM specifies an order to these Nodes.

`NamedNodeMap` objects in the DOM are live.

See also the Document Object Model (DOM) Level 3 Core Specification.

Namespace related methods are not executed .

Get clear in 5th round study

Method Summary

only these 2 we use

int	**getLength**() The number of nodes in this map.
Node	**getNamedItem**(java.lang.String name) Retrieves a node specified by name.
Node	**getNamedItemNS**(java.lang.String namespaceURI, java.lang.String localName) Retrieves a node specified by local name and namespace URI.
Node	**item**(int index) Returns the `index`th item in the map.

Node	**removeNamedItem**(java.lang.String name) Removes a node specified by name.
Node	**removeNamedItemNS**(java.lang.String namespaceURI, java.lang.String localName) Removes a node specified by local name and namespace URI.
Node	**setNamedItem**(Node arg) Adds a node using its nodeName attribute.
Node	**setNamedItemNS**(Node arg) Adds a node using its namespaceURI and localName.

Method Detail

getNamedItem

Node **getNamedItem**(java.lang.String name)

Retrieves a node specified by name.

Parameters:
name - The nodeName of a node to retrieve.

Returns:
A Node (of any type) with the specified nodeName, or null if it does not identify any node in this map.

---

setNamedItem

Node **setNamedItem**(Node arg)
throws DOMException

Adds a node using its nodeName attribute. If a node with that name is already present in this map, it is replaced by the new one. Replacing a node by itself has no effect.
As the nodeName attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a "special" string value) cannot be stored as the

names would clash. This is seen as preferable to allowing nodes to be aliased.

Parameters:

`arg` - A node to store in this map. The node will later be accessible using the value of its `nodeName` attribute.

Returns:

If the new `Node` replaces an existing node the replaced `Node` is returned, otherwise `null` is returned.

Throws:

`DOMException` - WRONG_DOCUMENT_ERR: Raised if `arg` was created from a different document than the one that created this map.
NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
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 re-use them in other elements.
HIERARCHY_REQUEST_ERR: Raised if an attempt is made to add a node doesn't belong in this NamedNodeMap. Examples would include trying to insert something other than an Attr node into an Element's map of attributes, or a non-Entity node into the DocumentType's map of Entities.

---

removeNamedItem

`Node` **`removeNamedItem`**`(java.lang.String name)`
`throws DOMException`

~~Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.~~

Parameters:

`name` - The `nodeName` of the node to remove.

Returns:

The node removed from this map if a node with such a name exists.

Throws:

`DOMException` - NOT_FOUND_ERR: Raised if there is no node named `name` in this map.
NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

---

item

`Node item(int index)`

Returns the `index`th item in the map. If `index` is greater than or equal to the number of nodes in this map, this returns `null`.

Parameters:

`index` - Index into this map.

Returns:

The node at the `index`th position in the map, or `null` if that is not a valid index.

---

getLength

`int getLength()`

The number of nodes in this map. The range of valid child node indices is `0` to `length-1` inclusive.

---

getNamedItemNS

```
Node getNamedItemNS(java.lang.String namespaceURI,
                    java.lang.String localName)
                    throws DOMException
```

Retrieves a node specified by local name and namespace URI.
Per [XML Namespaces], applications must use the value null as the namespaceURI parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the node to retrieve.

`localName` - The local name of the node to retrieve.

Returns:

A `Node` (of any type) with the specified local name and namespace URI, or `null` if they do not identify any node in this map.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature "XML" and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

setNamedItemNS

```
Node setNamedItemNS(Node arg)
                    throws DOMException
```

~~Adds a node using its `namespaceURI` and `localName`. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one. Replacing a node by itself has no effect.~~
~~Per [XML Namespaces] , applications must use the value null as the namespaceURI parameter for methods if they wish to have no namespace.~~

Parameters:

`arg` - A node to store in this map. The node will later be accessible using the value of its `namespaceURI` and `localName` attributes.

Returns:

If the new `Node` replaces an existing node the replaced `Node` is returned, otherwise `null` is returned.

Throws:

`DOMException` - WRONG_DOCUMENT_ERR: Raised if `arg` was created from a different document than the one that created this map.
NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
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 re-use them in other elements.
HIERARCHY_REQUEST_ERR: Raised if an attempt is made to add a node doesn't belong in this NamedNodeMap. Examples would include trying to insert something other than an Attr node into an Element's map of attributes, or a non-Entity node into the DocumentType's map of Entities.
NOT_SUPPORTED_ERR: May be raised if the implementation does not support the

feature "XML" and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

removeNamedItemNS

```
Node removeNamedItemNS(java.lang.String namespaceURI,
                       java.lang.String localName)
                       throws DOMException
```

Removes a node specified by local name and namespace URI. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the attributes attribute of the `Node` interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.
Per [XML Namespaces] , applications must use the value null as the namespaceURI parameter for methods if they wish to have no namespace.

Parameters:

`namespaceURI` - The namespace URI of the node to remove.

`localName` - The local name of the node to remove.

Returns:

The node removed from this map if a node with such a local name and namespace URI exists.

Throws:

`DOMException` - NOT_FOUND_ERR: Raised if there is no node with the specified `namespaceURI` and `localName` in this map.
NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.
NOT_SUPPORTED_ERR: May be raised if the implementation does not support the feature "XML" and the language exposed through the Document does not support XML Namespaces (such as [HTML 4.01]).

Since:

DOM Level 2

---

org.w3c.dom

Interface NameList

public interface **NameList**

The `NameList` interface provides the abstraction of an ordered collection of parallel pairs of name and namespace values (which could be null values), without defining or constraining how this collection is implemented. The items in the `NameList` are accessible via an integral index, starting from 0.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Method Summary

boolean	**contains**(java.lang.String str) Test if a name is part of this `NameList`.
boolean	**containsNS**(java.lang.String namespaceURI, java.lang.String name) Test if the pair namespaceURI/name is part of this `NameList`.
int	**getLength**() The number of pairs (name and namespaceURI) in the list.
java.lang.String	**getName**(int index) Returns the `index`th name item in the collection.

java.lang.String	**getNamespaceURI**(int index) Returns the indexth namespaceURI item in the collection.

Method Detail

getName

```
java.lang.String getName(int index)
```

Returns the indexth name item in the collection.

Parameters:

index - Index into the collection.

Returns:

The name at the indexth position in the NameList, or null if there is no name for the specified index or if the index is out of range.

---

getNamespaceURI

```
java.lang.String getNamespaceURI(int index)
```

Returns the indexth namespaceURI item in the collection.

Parameters:

index - Index into the collection.

Returns:

The namespace URI at the indexth position in the NameList, or null if there is no name for the specified index or if the index is out of range.

---

getLength

```
int getLength()
```

The `Iterator` is *not* modifiable. e.g. the `remove()` method will throw `UnsupportedOperationException`.

When requesting prefixes by Namespace URI, the following table describes the returned prefixes value for all Namespace URI values:

`getPrefixes(namespaceURI)` return value for specified Namespace URIs	
Namespace URI parameter	**prefixes value returned**
bound Namespace URI, including the <default Namespace URI>	`Iterator` over prefixes bound to Namespace URI in the current scope in an arbitrary, **implementation dependent**, order
unbound Namespace URI	empty `Iterator`
`XMLConstants.XML_NS_URI` ("http://www.w3.org/XML/1998/namespace")	`Iterator` with one element set to `XMLConstants.XML_NS_PREFIX` ("xml")
`XMLConstants.XMLNS_ATTRIBUTE_NS_URI` ("http://www.w3.org/2000/xmlns/")	`Iterator` with one element set to `XMLConstants.XMLNS_ATTRIBUTE` ("xmlns")
`null`	`IllegalArgumentException` is thrown

Parameters:

`namespaceURI` - URI of Namespace to lookup

Returns:

`Iterator` for all prefixes bound to the Namespace URI in the current scope

Throws:

`java.lang.IllegalArgumentException` - When `namespaceURI` is `null`

dont concentrate methods are related prefix, namespace, userdata . since those are mostly DOM implementation depdent

10 - OCT - 2008

o 15 - Dec - 08

Interface Node

it has 37 methods

8 - methods are child node related
20 - methods are related node itself
2 - methods are related to attribute
7 - methods are document / general
(SEE MY EXCEL)

All Known Subinterfaces:

Attr, CDATASection, CharacterData, Comment, Document, DocumentFragment, ~~DocumentType~~, Element, ~~Entity~~, ~~EntityReference~~, ~~Notation~~, ~~ProcessingInstruction~~, Text

8 - Sub interfaces

Most of methods would return NULL. so, while coding mostly need to null check before to access that returned object

```
public interface Node
```

The `Node` interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the `Node` interface expose methods for dealing with children, not all objects implementing the `Node` interface may have children. For example, `Text` nodes may not have children, and adding children to such nodes results in a `DOMException` being raised.

The attributes `nodeName`, `nodeValue` and `attributes` are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific `nodeType` (e.g., `nodeValue` for an `Element` or `attributes` for a `Comment`), this returns `null`. ~~Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.~~

The values of `nodeName`, `nodeValue`, and `attributes` vary according to the node type as follows:

Hard coded value

Interface	nodeName	nodeValue	attributes
`Attr`	same as `Attr.name`	same as `Attr.value`	`null`
`CDATASection`	`"#cdata-section"`	same as `CharacterData.data`, the content of the CDATA Section	`null`
`Comment`	`"#comment"`	same as `CharacterData.data`, the content of the comment	`null`
`Document`	`"#document"`	`null`	`null`
`DocumentFragment`	`"#document-fragment"`	`null`	`null`
~~`DocumentType`~~	~~same as `DocumentType.name`~~	~~`null`~~	~~`null`~~
`Element`	same as `Element.tagName`	`null` yes	`NamedNodeMap`

`Entity`	entity name	`null`	`null`
`EntityReference`	name of entity referenced	`null`	`null`
`Notation`	notation name	`null`	`null`
`ProcessingInstruction`	same as `ProcessingInstruction.target`	same as `ProcessingInstruction.data`	`null`
`Text`	`"#text"`	same as `CharacterData.data`, the content of the text node	`null`

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

static short	**`ATTRIBUTE_NODE`** The node is an `Attr`.
static short	**`CDATA_SECTION_NODE`** The node is a `CDATASection`.
static short	**`COMMENT_NODE`** The node is a `Comment`.
static short	**`DOCUMENT_FRAGMENT_NODE`** The node is a `DocumentFragment`.
static short	**`DOCUMENT_NODE`** The node is a `Document`.
static short	**`DOCUMENT_POSITION_CONTAINED_BY`** The node is contained by the reference node.
static short	**`DOCUMENT_POSITION_CONTAINS`** The node contains the reference node.
static short	**`DOCUMENT_POSITION_DISCONNECTED`** The two nodes are disconnected.
static short	**`DOCUMENT_POSITION_FOLLOWING`** The node follows the reference node.
static short	**`DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC`** The determination of preceding versus following is implementation-specific.
static short	**`DOCUMENT_POSITION_PRECEDING`** The second node precedes the reference node.

static short	**DOCUMENT_TYPE_NODE** The node is a DocumentType.
static short	**ELEMENT_NODE** The node is an Element.
static short	**ENTITY_NODE** The node is an Entity.
static short	**ENTITY_REFERENCE_NODE** The node is an EntityReference.
static short	**NOTATION_NODE** The node is a Notation.
static short	**PROCESSING_INSTRUCTION_NODE** The node is a ProcessingInstruction.
static short	**TEXT_NODE** The node is a Text node.

Method Summary

for child Node	**appendChild**(Node newChild) Adds the node newChild to the end of the list of children of this node.
Node itself Node	**cloneNode**(boolean deep) Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
short super functionality	**compareDocumentPosition**(Node other) Compares the reference node, i.e. the node on which this method is being called, with a node, i. e. the one passed as a parameter, with regard to their position in the document and according to the document order.
NamedNodeMap Attribute	**getAttributes**() A NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
java.lang. Node itself String	**getBaseURI**() The absolute base URI of this node or null if the implementation wasn't able to obtain an absolute URI.
NodeList for child	**getChildNodes**() A NodeList that contains all children of this node.
java.lang. Object	**getFeature**(java.lang.String feature, java.lang.String version) This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in .
for child Node	**getFirstChild**() The first child of this node.

it is same as Attributes of SAX

this method will return non-null value only if the documentbuilder factory enabled for namespace

Node for child	**getLastChild**() The last child of this node.
java.lang.String Node itself	**getLocalName**() Returns the local part of the qualified name of this node.
java.lang.String Node itself	**getNamespaceURI**() The namespace URI of this node, or `null` if it is unspecified (see).
Node sibling	**getNextSibling**() The node immediately following this node.
java.lang.String Node itself	**getNodeName**() The name of this node, depending on its type; see the table above.
short Node itself	**getNodeType**() A code representing the type of the underlying object, as defined above.
java.lang.String Node itself	**getNodeValue**() The value of this node, depending on its type; see the table above.
Document	**getOwnerDocument**() The `Document` object associated with this node.
Node	**getParentNode**() The parent of this node.
java.lang.String Node itself	**getPrefix**() The namespace prefix of this node, or `null` if it is unspecified.
Node sibling	**getPreviousSibling**() The node immediately preceding this node.
java.lang.String Node itself	**getTextContent**() This attribute returns the text content of this node and its descendants.
java.lang.Object Node itself	**getUserData**(java.lang.String key) Retrieves the object associated to a key on a this node.
boolean Attribute	**hasAttributes**() Returns whether this node (if it is an element) has any attributes.
boolean for child	**hasChildNodes**() Returns whether this node has any children.
Node for child	**insertBefore**(Node newChild, Node refChild) Inserts the node `newChild` before the existing child node `refChild`.
boolean Node itself	**isDefaultNamespace**(java.lang.String namespaceURI) This method checks if the specified `namespaceURI` is the default namespace or not.
boolean Node itself	**isEqualNode**(Node arg) Tests whether two nodes are equal.

boolean	**isSameNode**(Node other) Returns whether this node is the same node as the given one.
boolean	**isSupported**(java.lang.String feature, java.lang.String version) Tests whether the DOM implementation implements a specific feature and that feature is supported by this node, as specified in .
va.lang.String	**lookupNamespaceURI**(java.lang.String prefix) Look up the namespace URI associated to the given prefix, starting from this node.
va.lang.String	**lookupPrefix**(java.lang.String namespaceURI) Look up the prefix associated to the given namespace URI, starting from this node.
void	**normalize**() Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes, i.e., there are neither adjacent Text nodes nor empty Text nodes.
Node	**removeChild**(Node oldChild) Removes the child node indicated by oldChild from the list of children, and returns it.
Node	**replaceChild**(Node newChild, Node oldChild) Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.
void	**setNodeValue**(java.lang.String nodeValue) The value of this node, depending on its type; see the table above.
void	**setPrefix**(java.lang.String prefix) The namespace prefix of this node, or null if it is unspecified.
void	**setTextContent**(java.lang.String textContent) This attribute returns the text content of this node and its descendants.
java.lang.Object	**setUserData**(java.lang.String key, java.lang.Object data, UserDataHandler handler) Associate an object to a key on this node.

Field Detail

ELEMENT_NODE

```
static final short ELEMENT_NODE
```

The node is an Element.

See Also:

Constant Field Values

ATTRIBUTE_NODE

`static final short` **`ATTRIBUTE_NODE`**

The node is an `Attr`.

See Also:
Constant Field Values

TEXT_NODE

`static final short` **`TEXT_NODE`**

The node is a `Text` node.

See Also:
Constant Field Values

CDATA_SECTION_NODE

`static final short` **`CDATA_SECTION_NODE`**

The node is a `CDATASection`.

See Also:
Constant Field Values

ENTITY_REFERENCE_NODE

`static final short` **`ENTITY_REFERENCE_NODE`**

The node is an `EntityReference`.

See Also:
Constant Field Values

ENTITY_NODE

```
static final short ENTITY_NODE
```

The node is an `Entity`.

See Also:
Constant Field Values

PROCESSING_INSTRUCTION_NODE

```
static final short PROCESSING_INSTRUCTION_NODE
```

The node is a `ProcessingInstruction`.

See Also:
Constant Field Values

COMMENT_NODE

```
static final short COMMENT_NODE
```

The node is a `Comment`.

See Also:
Constant Field Values

DOCUMENT_NODE

```
static final short DOCUMENT_NODE
```

The node is a `Document`.

See Also:
Constant Field Values

DOCUMENT_TYPE_NODE

`static final short` **`DOCUMENT_TYPE_NODE`**

The node is a `DocumentType`.

See Also:
Constant Field Values

DOCUMENT_FRAGMENT_NODE

`static final short` **`DOCUMENT_FRAGMENT_NODE`**

The node is a `DocumentFragment`.

See Also:
Constant Field Values

NOTATION_NODE

`static final short` **`NOTATION_NODE`**

The node is a `Notation`.

See Also:
Constant Field Values

DOCUMENT_POSITION_DISCONNECTED

`static final short` **`DOCUMENT_POSITION_DISCONNECTED`**

The two nodes are disconnected. Order between disconnected nodes is always implementation-specific.

See Also:
Constant Field Values

DOCUMENT_POSITION_PRECEDING

`static final short` **`DOCUMENT_POSITION_PRECEDING`**

The second node precedes the reference node.

See Also:
Constant Field Values

DOCUMENT_POSITION_FOLLOWING

`static final short` **`DOCUMENT_POSITION_FOLLOWING`**

The node follows the reference node.

See Also:
Constant Field Values

DOCUMENT_POSITION_CONTAINS

`static final short` **`DOCUMENT_POSITION_CONTAINS`**

The node contains the reference node. A node which contains is always preceding, too.

See Also:
Constant Field Values

DOCUMENT_POSITION_CONTAINED_BY

`static final short` **`DOCUMENT_POSITION_CONTAINED_BY`**

The node is contained by the reference node. A node which is contained is always following, too.

See Also:
Constant Field Values

DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC

`static final short` **`DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC`**

The determination of preceding versus following is implementation-specific.

See Also:
Constant Field Values

Method Detail

getNodeName

```
java.lang.String getNodeName()
```

it never return null

The name of this node, depending on its type; see the table above.

---

getNodeValue

yes, it is correct

it return NULL always for element node type

```
java.lang.String getNodeValue()
                          throws DOMException
```

The value of this node, depending on its type; see the table above. When it is defined to be `null`, setting it has no effect, including if the node is read-only.

Throws:

DOMException - DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

---

setNodeValue

```
void setNodeValue(java.lang.String nodeValue)
                  throws DOMException
```

The value of this node, depending on its type; see the table above. When it is defined to be `null`, setting it has no effect, including if the node is read-only.

Throws:

DOMException - NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly and if it is not defined to be `null`.

---

getNodeType

```
short getNodeType()
```

A code representing the type of the underlying object, as defined above.

---

getParentNode

`Node` **`getParentNode()`**

yes. i tested

The parent of this node. All nodes, except `Attr`, `Document`, `DocumentFragment`, `Entity`, and `Notation` 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`.

---

getChildNode

Never return null but may be EMPTY object

it will return only child nodes but not grand child nodes. so to get grand child nodes again has to use the same methods on child nodes

`NodeList` **`getChildNodes()`**

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

---

getFirstChild

`Node` **`getFirstChild()`**

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

---

getLastChild

Need Nulll Check

`Node` **`getLastChild()`**

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

---

getPreviousSibling

`Node` **`getPreviousSibling()`**

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

---

getNextSibling

`Node` **`getNextSibling()`**

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

---

getAttributes

`NamedNodeMap` **`getAttributes()`**

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

---

getOwnerDocument

`Document` **`getOwnerDocument()`**

The `Document` object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document` or a `DocumentType` which is not used with any `Document` yet, this is `null`.

Since:
DOM Level 2

so, only newly created document possible to return null (i tested)

---

insertBefore

```
Node insertBefore(Node newChild,
                  Node refChild)
                  throws DOMException
```

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

Note: Inserting a node before itself is implementation dependent.

here is null.

i tried, but got exception. since it is DOM level 3

Parameters:
`newChild` - The node to insert.
`refChild` - The reference node, i.e., the node before which the new node must be inserted.
Returns:
The node being inserted.
Throws:
`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 or this node itself, or if this node is of type `Document` and the DOM application attempts to insert a second `DocumentType` or `Element` node.
WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or if the parent of the node being inserted is readonly.
NOT_FOUND_ERR: Raised if `refChild` is not a child of this node.
NOT_SUPPORTED_ERR: if this node is of type `Document`, this exception might be raised if the DOM implementation doesn't support the insertion of a `DocumentType` or `Element` node.

Since:

DOM Level 3

---

replaceChild

```
Node replaceChild(Node newChild,
                  Node oldChild)
                  throws DOMException
```

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node. If `newChild` is a `DocumentFragment` object, `oldChild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newChild` is already in the tree, it is first removed.

Note: Replacing a node with itself is implementation dependent.

cannot replace itself. i tested

Parameters:

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

Returns:

The node replaced.

~~Throws:~~

`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 or this node itself, or if this node is of type `Document` and the result of the replacement operation would add a second `DocumentType` or `Element` on the `Document` node.~~
~~WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.~~
~~NO_MODIFICATION_ALLOWED_ERR: Raised if this node or the parent of the new node is readonly.~~
~~NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.~~
~~NOT_SUPPORTED_ERR: if this node is of type `Document`, this exception might be raised if the DOM implementation doesn't support the replacement of the `DocumentType` child or `Element` child.~~

Since:

DOM Level 3

---

removeChild

```
Node removeChild(Node oldChild)
                 throws DOMException
```

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

Parameters:

`oldChild` - The node being removed.

Returns:

The node removed.

Throws:

DOMException - NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.
NOT_SUPPORTED_ERR: if this node is of type `Document`, this exception might be raised if the DOM implementation doesn't support the removal of the `DocumentType` child or the `Element` child.

Since:

DOM Level 3

---

appendChild

```
Node appendChild(Node newChild)
                 throws DOMException
```

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:

`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:

The node added.

Throws:

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 or this node itself, or if this node is of type `Document` and the DOM application attempts to append a second `DocumentType` or `Element` node.
WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.
NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or if the previous parent of the node being inserted is readonly.
NOT_SUPPORTED_ERR: if the `newChild` node is a child of the `Document` node, this exception might be raised if the DOM implementation doesn't support the removal of the `DocumentType` child or `Element` child.

Since:

DOM Level 3

---

hasChildNodes

`boolean` **`hasChildNodes`**`()`

Returns whether this node has any children.

Returns:

Returns `true` if this node has any children, `false` otherwise.

---

cloneNode

`Node` **`cloneNode`**`(boolean deep)`

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` is `null`) and no user data. User data associated to the imported node is not carried over. However, if any `UserDataHandlers` has been specified along with the associated data these handlers will be called with the appropriate parameters before this method returns.
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 children it contains unless it is a deep clone. This includes text contained in an the `Element` since the text is contained in a child `Text` node. Cloning an `Attr` directly, as opposed to be cloned as part of an `Element` cloning operation, returns a specified attribute (`specified` is `true`). Cloning an `Attr` always clones its children, since they represent its value, no matter whether this is a deep clone or not. Cloning an `EntityReference` automatically constructs its subtree if a corresponding `Entity` is available, no matter whether this is a deep clone or not. Cloning any other type of node simply returns a copy of this node.
Note that cloning an immutable subtree results in a mutable copy, but the children of an `EntityReference` clone are readonly . In addition, clones of unspecified `Attr` nodes are specified. And, cloning `Document`, `DocumentType`, `Entity`, and `Notation` nodes is implementation dependent.

Parameters:

`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:

The duplicate node.

---

normalize

`void` **`normalize`**`()`

this method has highly impact only on XPointer. i hope, i dont want XPointer

Puts all `Text` nodes in the full depth of the sub-tree underneath this `Node`, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are neither adjacent `Text` nodes nor empty `Text` nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as XPointer [XPointer] lookups) that depend on a particular document tree structure are to be used. If the parameter "normalize-characters" of the `DOMConfiguration` object attached to the `Node.ownerDocument` is `true`, this method will also fully normalize the characters of the `Text` nodes.

Note: In cases where the document contains `CDATASections`, the normalize operation alone may not be sufficient, since XPointers do not differentiate between `Text` nodes and `CDATASection` nodes.

Since:

DOM Level 3

---

isSupported

```
boolean isSupported(java.lang.String feature,
                    java.lang.String version)
```

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node, as specified in .

Parameters:

`feature` - The name of the feature to test.

`version` - This is the version number of the feature to test.

Returns:

Returns `true` if the specified feature is supported on this node, `false` otherwise.

Since:

DOM Level 2

---

getNamespaceURI

```
java.lang.String getNamespaceURI()
```

The namespace URI of this node, or `null` if it is unspecified (see).

This is not a computed value that is the result of a namespace lookup based on an examination of the namespace declarations in scope. It is merely the namespace URI given at creation time.

For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `Document.createElement()`, this is always `null`.

Note: Per the *Namespaces in XML* Specification [XML Namespaces] an attribute does not inherit its namespace from the element it is attached to. If an attribute is not explicitly given a namespace, it simply has no namespace.

only prefix ??

Since:

DOM Level 2

---

getPrefix

```
java.lang.String getPrefix()
```

on current node

The namespace prefix of this node, or `null` if it is unspecified. When it is defined to be `null`, setting it has no effect, including if the node is read-only.
Note that setting this attribute, when permitted, changes the `nodeName` attribute, which holds the qualified name, as well as the `tagName` and `name` attributes of the `Element` and `Attr` interfaces, when applicable.
Setting the prefix to `null` makes it unspecified, setting it to an empty string is implementation dependent.
Note also that changing the prefix of an attribute that is known to have a default value, does not make a new attribute with the default value and the original prefix appear, since the `namespaceURI` and `localName` do not change.
For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` interface, this is always `null`.

since is has to follow the same prefix of element

Since:

DOM Level 2

---

setPrefix

```
void setPrefix(java.lang.String prefix)
               throws DOMException
```

~~The namespace prefix of this node, or `null` if it is unspecified. When it is defined to be `null`, setting it has no effect, including if the node is read-only.~~
~~Note that setting this attribute, when permitted, changes the `nodeName` attribute, which holds the qualified name, as well as the `tagName` and `name` attributes of the `Element` and `Attr` interfaces, when applicable.~~
~~Setting the prefix to `null` makes it unspecified, setting it to an empty string is implementation dependent.~~
~~Note also that changing the prefix of an attribute that is known to have a default value, does not make a new attribute with the default value and the original prefix appear, since the `namespaceURI` and `localName` do not change.~~
~~For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` interface, this is always `null`.~~

Throws:

`DOMException` - ~~INVALID_CHARACTER_ERR: Raised if the specified prefix contains an illegal character according to the XML version in use specified in the `Document.xmlVersion` attribute.~~
~~NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.~~
NAMESPACE_ERR: Raised if the specified `prefix` is malformed per the Namespaces in XML

specification, if the `namespaceURI` of this node is `null`, if the specified prefix is "xml" and the `namespaceURI` of this node is different from " http://www.w3.org/XML/1998/namespace", if this node is an attribute and the specified prefix is "xmlns" and the `namespaceURI` of this node is different from "http://www.w3.org/2000/xmlns/", or if this node is an attribute and the `qualifiedName` of this node is "xmlns" [XML Namespaces] .

Since:

DOM Level 2

---

getLocalName

`java.lang.String` **`getLocalName`**`()`

Returns the local part of the qualified name of this node.
For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `Document.createElement()`, this is always `null`.

this method will return non-null value only if the documentbuilder factory enabled for namespace

Since:

DOM Level 2

---

hasAttributes

`boolean` **`hasAttributes`**`()`

even Namespace declaration also treated as Attribute of that particular element

~~Returns whether this node (if it is an element) has any attributes.~~

~~**Returns:**~~

~~Returns `true` if this node has any attributes, `false` otherwise.~~

~~**Since:**~~

~~DOM Level 2~~

---

getBaseURI

it did not work. it might be because of DOM Level 3

`java.lang.String` **`getBaseURI`**`()`

The absolute base URI of this node or `null` if the implementation wasn't able to obtain an absolute URI. This value is computed as described in . ~~However, when the `Document` supports the feature "HTML" [DOM Level 2 HTML] , the base URI is computed using first the value of the href attribute of the HTML BASE element if any, and the value of the `documentURI` attribute from the `Document` interface otherwise.~~

Since:

DOM Level 3

---

compareDocumentPosition

```
short compareDocumentPosition(Node other)
                              throws DOMException
```

i dont know exactly what the return value rep. but executed successfully

Compares the reference node, i.e. the node on which this method is being called, with a node, i.e. the one passed as a parameter, with regard to their position in the document and according to the document order.

Parameters:

`other` - The node to compare against the reference node.

Returns:

Returns how the node is positioned relatively to the reference node.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: when the compared nodes are from different DOM implementations that do not coordinate to return consistent implementation-specific results.

Since:

DOM Level 3

---

getTextContent

```
java.lang.String getTextContent()
                              throws DOMException
```

yes

This attribute returns the text content of this node and its descendants. When it is defined to be `null`, setting it has no effect. On setting, any possible children this node may have are removed and, if it the new string is not empty or `null`, replaced by a single `Text` node containing the string this attribute is set to.

On getting, no serialization is performed, the returned string does not contain any markup. No whitespace normalization is performed and the returned string does not contain the white spaces in element content (see the attribute `Text.isElementContentWhitespace`). Similarly, on setting, no parsing is performed either, the input string is taken as pure textual content.

The string returned is made of the text content of this node depending on its type, as defined below:

Node type	Content
ELEMENT_NODE, ATTRIBUTE_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, DOCUMENT_FRAGMENT_NODE	concatenation of the `textContent` attribute value of every child node, excluding COMMENT_NODE and PROCESSING_INSTRUCTION_NODE nodes. This is the empty string if the node has no children.
TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE, PROCESSING_INSTRUCTION_NODE	`nodeValue`
DOCUMENT_NODE, DOCUMENT_TYPE_NODE, NOTATION_NODE	*null*

Throws:

DOMException - DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

Since:

DOM Level 3

---

setTextContent

```
void setTextContent(java.lang.String textContent)
                    throws DOMException
```

This attribute returns the text content of this node and its descendants. When it is defined to be `null`, setting it has no effect. On setting, any possible children this node may have are removed and, if it the new string is not empty or `null`, replaced by a single `Text` node containing the string this attribute is set to.

On getting, no serialization is performed, the returned string does not contain any markup. No whitespace normalization is performed and the returned string does not contain the white spaces in element content (see the attribute `Text.isElementContentWhitespace`). Similarly, on setting, no parsing is performed either, the input string is taken as pure textual content.

The string returned is made of the text content of this node depending on its type, as defined below:

Node type	Content
ELEMENT_NODE, ATTRIBUTE_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, DOCUMENT_FRAGMENT_NODE	concatenation of the `textContent` attribute value of every child node, excluding COMMENT_NODE and PROCESSING_INSTRUCTION_NODE nodes. This is the empty string if the node has no children.
TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE, PROCESSING_INSTRUCTION_NODE	`nodeValue`
DOCUMENT_NODE, DOCUMENT_TYPE_NODE, NOTATION_NODE	*null*

Throws:

DOMException - NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Since:

DOM Level 3

---

isSameNode

```
boolean isSameNode(Node other)
```

Returns whether this node is the same node as the given one.

This method provides a way to determine whether two `Node` references returned by the implementation reference

the same object. When two `Node` references are references to the same object, even if through a proxy, the references may be used completely interchangeably, such that all attributes have the same values and calling the same DOM method on either reference always has exactly the same effect.

Parameters:
`other` - The node to test against.

Returns:
Returns `true` if the nodes are the same, `false` otherwise.

Since:
DOM Level 3

---

lookupPrefix

```
java.lang.String lookupPrefix(java.lang.String namespaceURI)
```

Look up the prefix associated to the given namespace URI, starting from this node. The default namespace declarations are ignored by this method.
See for details on the algorithm used by this method.

i got null, even for non-default namespace

Parameters:
`namespaceURI` - The namespace URI to look for.

Returns:
Returns an associated namespace prefix if found or `null` if none is found. If more than one prefix are associated to the namespace prefix, the returned namespace prefix is implementation dependent.

Since:
DOM Level 3

---

isDefaultNamespace

```
boolean isDefaultNamespace(java.lang.String namespaceURI)
```

This method checks if the specified `namespaceURI` is the default namespace or not.

Super

Parameters:
`namespaceURI` - The namespace URI to look for.

Returns:
Returns `true` if the specified `namespaceURI` is the default namespace, `false` otherwise.

Since:
DOM Level 3

---

lookupNamespaceURI

`java.lang.String` **`lookupNamespaceURI`**`(java.lang.String prefix)`

Look up the namespace URI associated to the given prefix, starting from this node.
See for details on the algorithm used by this method.

Parameters:

`prefix` - The prefix to look for. If this parameter is `null`, the method will return the default namespace URI if any.

Returns:

Returns the associated namespace URI or `null` if none is found.

Since:

DOM Level 3

---

isEqualNode

`boolean` **`isEqualNode`**`(`Node `arg)`

will it consider child nodes equality or only consider only those two element

Tests whether two nodes are equal.
This method tests for equality of nodes, not sameness (i.e., whether the two nodes are references to the same object) which can be tested with `Node.isSameNode()`. All nodes that are the same will also be equal, though the reverse may not be true.
Two nodes are equal if and only if the following conditions are satisfied:

- The two nodes are of the same type.
- The following string attributes are equal: `nodeName`, `localName`, `namespaceURI`, `prefix`, `nodeValue` . This is: they are both `null`, or they have the same length and are character for character identical.
- The `attributes NamedNodeMaps` are equal. This is: they are both `null`, or they have the same length and for each node that exists in one map there is a node that exists in the other map and is equal, although not necessarily at the same index.
- The `childNodes NodeLists` are equal. This is: they are both `null`, or they have the same length and contain equal nodes at the same index. Note that normalization can affect equality; to avoid this, nodes should be normalized before being compared.

For two `DocumentType` nodes to be equal, the following conditions must also be satisfied:

- The following string attributes are equal: `publicId`, `systemId`, `internalSubset`.
- The `entities NamedNodeMaps` are equal.
- The `notations NamedNodeMaps` are equal.

On the other hand, the following do not affect equality: the `ownerDocument`, `baseURI`, and `parentNode` attributes, the `specified` attribute for `Attr` nodes, the `schemaTypeInfo` attribute for `Attr` and `Element` nodes, the `Text.isElementContentWhitespace` attribute for `Text` nodes, as well as any user data or event listeners registered on the nodes.

Note: As a general rule, anything not mentioned in the description above is not significant in consideration of equality checking. Note that future versions of this specification may take into account more attributes and implementations conform to this specification are expected to be updated accordingly.

Parameters:

`arg` - The node to compare equality with.

Returns:

Returns `true` if the nodes are equal, `false` otherwise.

Since:

DOM Level 3

---

getFeature

```
java.lang.Object getFeature(java.lang.String feature,
                            java.lang.String version)
```

This method returns a specialized object which implements the specialized APIs of the specified feature and version, as specified in . The specialized object may also be obtained by using binding-specific casting methods but is not necessarily expected to, as discussed in . This method also allow the implementation to provide specialized objects which do not support the `Node` interface.

Parameters:

`feature` - The name of the feature requested. Note that any plus sign "+" prepended to the name of the feature will be ignored since it is not significant in the context of this method.

`version` - This is the version number of the feature to test.

Returns:

Returns an object which implements the specialized APIs of the specified feature and version, if any, or `null` if there is no object which implements interfaces associated with that feature. If the `DOMObject` returned by this method implements the `Node` interface, it must delegate to the primary core `Node` and not return results inconsistent with the primary core `Node` such as attributes, childNodes, etc.

Since:

DOM Level 3

---

setUserData

so, associate userData handler is optional.

```
java.lang.Object setUserData(java.lang.String key,
                             java.lang.Object data,
                             UserDataHandler handler)
```

Associate an object to a key on this node. The object can later be retrieved from this node by calling `getUserData` with the same key.

Parameters:

`key` - The key to associate the object to.

`data` - The object to associate to the given key, or `null` to remove any existing association to that key.

`handler` - The handler to associate to that key, or `null`.

Returns:

Returns the `DOMUserData` previously associated to the given key on this node, or `null` if there was none.

Since:

DOM Level 3

---

getUserData

```
java.lang.Object getUserData(java.lang.String key)
```

Retrieves the object associated to a key on a this node. The object must first have been set to this node by calling `setUserData` with the same key.

Parameters:

`key` - The key the object is associated to.

Returns:

Returns the `DOMUserData` associated to the given key on this node, or `null` if there was none.

Since:

DOM Level 3

---

org.w3c.dom.traversal

Interface NodeFilter

only 1 methods but many variables

All Known Subinterfaces:

LSSerializerFilter

WhatToShow is rep. what type of node has to apply for filter. remaining type of node processed automatically without appling filter

---

`public interface NodeFilter`

~~Filters are objects that know how to "filter out" nodes. If a `NodeIterator` or `TreeWalker` is given a `NodeFilter`, it applies the filter before it returns the next node. If the filter says to accept the node, the traversal logic returns it; otherwise, traversal looks for the next node and pretends that the node that was rejected was not there.~~

The DOM does not provide any filters. `NodeFilter` is just an interface that users can implement to provide their own filters.

`NodeFilters` do not need to know how to traverse from node to node, nor do they need to know anything about the data structure that is being traversed. This makes it very easy to write filters, since the only thing they have to know how to do is evaluate a single node. One filter may be used with a number of different kinds of traversals, encouraging code reuse.

See also the Document Object Model (DOM) Level 2 Traversal and Range Specification.

Super design, the way of use of filter and its abstraction level

Since:

DOM Level 2

Field Summary

`static short`	**`FILTER_ACCEPT`** Accept the node.

static short	**FILTER_REJECT** Reject the node.
static short	**FILTER_SKIP** Skip this single node.
static int	**SHOW_ALL** Show all Nodes.
static int	**SHOW_ATTRIBUTE** Show Attr nodes.
static int	**SHOW_CDATA_SECTION** Show CDATASection nodes.
static int	**SHOW_COMMENT** Show Comment nodes.
static int	**SHOW_DOCUMENT** Show Document nodes.
static int	**SHOW_DOCUMENT_FRAGMENT** Show DocumentFragment nodes.
static int	**SHOW_DOCUMENT_TYPE** Show DocumentType nodes.
static int	**SHOW_ELEMENT** Show Element nodes.
static int	**SHOW_ENTITY** Show Entity nodes.
static int	**SHOW_ENTITY_REFERENCE** Show EntityReference nodes.
static int	**SHOW_NOTATION** Show Notation nodes.
static int	**SHOW_PROCESSING_INSTRUCTION** Show ProcessingInstruction nodes.
static int	**SHOW_TEXT** Show Text nodes.

Method Summary

short	**acceptNode**(Node n) Test whether a specified node is visible in the logical view of a TreeWalker or NodeIterator.

Field Detail

FILTER_ACCEPT

```
static final short FILTER_ACCEPT
```

Accept the node. Navigation methods defined for NodeIterator or TreeWalker will return this node.

See Also:

Constant Field Values

FILTER_REJECT

```
static final short FILTER_REJECT
```

Reject the node. Navigation methods defined for NodeIterator or TreeWalker will not return this node. For TreeWalker, the children of this node will also be rejected. NodeIterators treat this as a synonym for FILTER_SKIP.

See Also:

Constant Field Values

FILTER_SKIP

```
static final short FILTER_SKIP
```

Skip this single node. Navigation methods defined for `NodeIterator` or `TreeWalker` will not return this node. For both `NodeIterator` and `TreeWalker`, the children of this node will still be considered.

See Also:

Constant Field Values

SHOW_ALL

`static final int` **`SHOW_ALL`**

Show all `Nodes`.

See Also:

Constant Field Values

SHOW_ELEMENT

`static final int` **`SHOW_ELEMENT`**

Show `Element` nodes.

See Also:

Constant Field Values

SHOW_ATTRIBUTE

`static final int` **`SHOW_ATTRIBUTE`**

Show `Attr` nodes. This is meaningful only when creating an `NodeIterator` or `TreeWalker` with an attribute node as its `root`; in this case, it means that the attribute node will appear in the first position of the iteration or traversal. Since attributes are never children of other nodes, they do not appear when traversing over the document tree.

See Also:
Constant Field Values

SHOW_TEXT

```
static final int SHOW_TEXT
```

Show `Text` nodes.

See Also:
Constant Field Values

SHOW_CDATA_SECTION

```
static final int SHOW_CDATA_SECTION
```

Show `CDATASection` nodes.

See Also:
Constant Field Values

SHOW_ENTITY_REFERENCE

```
static final int SHOW_ENTITY_REFERENCE
```

Show `EntityReference` nodes.

See Also:
Constant Field Values

SHOW_ENTITY

`static final int` **`SHOW_ENTITY`**

Show `Entity` nodes. This is meaningful only when creating an `NodeIterator` or `TreeWalker` with an `Entity` node as its `root`; in this case, it means that the `Entity` node will appear in the first position of the traversal. Since entities are not part of the document tree, they do not appear when traversing over the document tree.

See Also:
Constant Field Values

SHOW_PROCESSING_INSTRUCTION

`static final int` **`SHOW_PROCESSING_INSTRUCTION`**

Show `ProcessingInstruction` nodes.

See Also:
Constant Field Values

SHOW_COMMENT

`static final int` **`SHOW_COMMENT`**

Show `Comment` nodes.

See Also:
Constant Field Values

SHOW_DOCUMENT

`static final int` **`SHOW_DOCUMENT`**

Show `Document` nodes.

See Also:

Constant Field Values

SHOW_DOCUMENT_TYPE

```
static final int SHOW_DOCUMENT_TYPE
```

Show `DocumentType` nodes.

See Also:

Constant Field Values

SHOW_DOCUMENT_FRAGMENT

```
static final int SHOW_DOCUMENT_FRAGMENT
```

Show `DocumentFragment` nodes.

See Also:

Constant Field Values

SHOW_NOTATION

```
static final int SHOW_NOTATION
```

Show `Notation` nodes. This is meaningful only when creating an `NodeIterator` or `TreeWalker` with a `Notation` node as its `root`; in this case, it means that the `Notation` node will appear in the first position of the traversal. Since notations are not part of the document tree, they do not appear when traversing over the document tree.

See Also:

Constant Field Values

Method Detail

acceptNode

`short` **`acceptNode`**`(Node n)`

Test whether a specified node is visible in the logical view of a `TreeWalker` or `NodeIterator`. This function will be called by the implementation of `TreeWalker` and `NodeIterator`; it is not normally called directly from user code. (Though you could do so if you wanted to use the same filter to guide your own application logic.)

Parameters:

`n` - The node to check to see if it passes the filter or not.

Returns:

A constant to determine whether the node is accepted, rejected, or skipped, as defined above.

---

21 - Oct - 08

Tree walker can be ONLY used with SubTree.
NodeIterator can be subtree / nodelist /xpath query result

19 - Dec - 08

it can be used to iterate

1. a NodeList object
2. subtree of a node
3. result of a query

org.w3c.dom.traversal

Interface NodeIterator

7 methods

3 - getter for parameters
1 - detach
2 - retrieve nodes either way

```
nodeIterator = documentTraversal.createNodeIterator(nodeList.item(0), 1, null, false);
```

```
public interface NodeIterator
```

`NodeIterators` are used to step through a set of nodes, e.g. the set of nodes in a `NodeList`, the document subtree governed by a particular `Node`, the results of a query, or any other set of nodes. The set of nodes to be iterated is determined by the implementation of the `NodeIterator`. DOM Level 2 specifies a single `NodeIterator` implementation for document-order traversal of a document subtree. Instances of these `NodeIterators` are created by calling `DocumentTraversal.createNodeIterator()`.

See also the Document Object Model (DOM) Level 2 Traversal and Range Specification.

Since:

DOM Level 2

Method Summary

void	**detach**() Detaches the `NodeIterator` from the set which it iterated over, releasing any computational resources and placing the `NodeIterator` in the INVALID state.
boolean	**getExpandEntityReferences**() The value of this flag determines whether the children of entity reference nodes are visible to the `NodeIterator`.
NodeFilter	**getFilter**() The `NodeFilter` used to screen nodes.

Node	**getRoot**() The root node of the NodeIterator, as specified when it was created.
int	**getWhatToShow**() This attribute determines which node types are presented via the NodeIterator.
Node	**nextNode**() Returns the next node in the set and advances the position of the NodeIterator in the set.
Node	**previousNode**() Returns the previous node in the set and moves the position of the NodeIterator backwards in the set.

Method Detail

getRoot

Node **getRoot**()

The root node of the NodeIterator, as specified when it was created.

---

filter actually works only on what to shows node types.

getWhatToShow

int **getWhatToShow**()

This attribute determines which node types are presented via the NodeIterator. The available set of constants is defined in the NodeFilter interface. Nodes not accepted by whatToShow will be skipped, but their children may still be considered. Note that this skip takes precedence over the filter, if any.

---

getFilter

`NodeFilter` **`getFilter()`**

The `NodeFilter` used to screen nodes.

---

getExpandEntityReferences

`boolean` **`getExpandEntityReferences()`**

The value of this flag determines whether the children of entity reference nodes are visible to the `NodeIterator`. If false, these children and their descendants will be rejected. Note that this rejection takes precedence over `whatToShow` and the filter. Also note that this is currently the only situation where `NodeIterators` may reject a complete subtree rather than skipping individual nodes.

To produce a view of the document that has entity references expanded and does not expose the entity reference node itself, use the `whatToShow` flags to hide the entity reference node and set `expandEntityReferences` to true when creating the `NodeIterator`. To produce a view of the document that has entity reference nodes but no entity expansion, use the `whatToShow` flags to show the entity reference node and set `expandEntityReferences` to false.

---

nextNode

`Node` **`nextNode()`**
`throws DOMException`

Returns the next node in the set and advances the position of the `NodeIterator` in the set. After a `NodeIterator` is created, the first call to `nextNode()` returns the first node in the set.

Returns:

The next `Node` in the set being iterated over, or `null` if there are no more members in that set.

Throws:

`DOMException` - INVALID_STATE_ERR: Raised if this method is called after the `detach` method was invoked.

yes.. it is raised

---

previousNode

`Node` **`previousNode`**`()`
`throws` `DOMException`

Returns the previous node in the set and moves the position of the `NodeIterator` backwards in the set.

Returns:

The previous `Node` in the set being iterated over, or `null` if there are no more members in that set.

Throws:

`DOMException` - INVALID_STATE_ERR: Raised if this method is called after the `detach` method was invoked.

---

detach

`void` **`detach`**`()`

Detaches the `NodeIterator` from the set which it iterated over, releasing any computational resources and placing the `NodeIterator` in the INVALID state. After `detach` has been invoked, calls to `nextNode` or `previousNode` will raise the exception INVALID_STATE_ERR.

---

18 - Dec - 08

it has only 2 methods

so the DOM has two collection for holding Node

1. NodeList
2. NamedNodeMap

org.w3c.dom

Interface NodeList

This is COLLECTION based LIST for holding Node

`public interface NodeList`

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. `NodeList` objects in the DOM are live.

The items in the `NodeList` are accessible via an integral index, starting from 0.

See also the Document Object Model (DOM) Level 3 Core Specification.

Method Summary

int	**getLength**() The number of nodes in the list.
Node	**item**(int index) Returns the `index`th item in the collection.

Method Detail

item

`Node item(int index)`

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of nodes in the list, this returns `null`.

NO Exception !!!

18 -Dec - 08

org.w3c.dom

Interface Text

it has 4 methods

All Superinterfaces:

CharacterData, Node

All Known Subinterfaces:

CDATASection

`public interface Text`

only these two can have Text as child node

extends CharacterData

The `Text` interface inherits from `CharacterData` and represents the textual content (termed character data in XML) of an `Element` or `Attr`. If there is no markup inside an element's content, the text is contained in a single object implementing the `Text` interface that is the only child of the element. If there is markup, it is parsed into the information items (elements, comments, etc.) and `Text` nodes that form the list of children of the element.

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 represent 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 `Node.normalize()` method merges any such adjacent `Text` objects into a single node for each block of text.

but, not merge CDATA node

~~No lexical check is done on the content of a `Text` node and, depending on its position in the document, some characters must be escaped during serialization using character references; e.g. the characters "<&" if the textual content is part of an element or of an attribute, the character sequence "]]>" when part of an element, the quotation mark character " or the apostrophe character ' when part of an attribute.~~

See also the Document Object Model (DOM) Level 3 Core Specification.

Field Summary

Fields inherited from interface org.w3c.dom.Node

ATTRIBUTE_NODE, CDATA_SECTION_NODE, COMMENT_NODE, DOCUMENT_FRAGMENT_NODE, DOCUMENT_NODE, DOCUMENT_POSITION_CONTAINED_BY, DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_DISCONNECTED, DOCUMENT_POSITION_FOLLOWING, DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC, DOCUMENT_POSITION_PRECEDING, DOCUMENT_TYPE_NODE, ELEMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, TEXT_NODE

Method Summary

java.lang.String	**getWholeText**() Returns all text of Text nodes logically-adjacent text nodes to this node, concatenated in document order.
boolean	**isElementContentWhitespace**() Returns whether this text node contains element content whitespace, often abusively called "ignorable whitespace".
Text	**replaceWholeText**(java.lang.String content) Replaces the text of the current node and all logically-adjacent text nodes with the specified text.
Text	**splitText**(int offset) Breaks this node into two nodes at the specified offset, keeping both in the tree as siblings.

Methods inherited from interface org.w3c.dom.CharacterData

appendData, deleteData, getData, getLength, insertData, replaceData, setData, substringData

Methods inherited from interface org.w3c.dom.Node

appendChild, cloneNode, compareDocumentPosition, getAttributes, getBaseURI, getChildNodes, getFeature, getFirstChild, getLastChild, getLocalName, getNamespaceURI, getNextSibling, getNodeName, getNodeType, getNodeValue, getOwnerDocument, getParentNode, getPrefix, getPreviousSibling, getTextContent, getUserData, hasAttributes, hasChildNodes, insertBefore, isDefaultNamespace, isEqualNode, isSameNode, isSupported, lookupNamespaceURI, lookupPrefix, normalize, removeChild, replaceChild, setNodeValue, setPrefix, setTextContent, setUserData

Method Detail

splitText

it gets copied the whole text only but not from specified index.

```
Text splitText(int offset)
               throws DOMException
```

this is newly created Text node after offset

Breaks this node into two nodes at the specified `offset`, keeping both in the tree as siblings. After being split, this node will contain all the content up to the `offset` point. A new node of the same type, which contains all the content at and after the `offset` point, is returned. If the original node had a parent node, the new node is inserted as the next sibling of the original node. When the `offset` is equal to the length of this node, the new node has no data.

Parameters:

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

Returns:

The new node, of the same type as this node.

Throws:

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

---

isElementContentWhitespace

```
boolean isElementContentWhitespace()
```

Returns whether this text node contains element content whitespace, often abusively called "ignorable whitespace". The text node is determined to contain whitespace in element content during the load of the document or if validation occurs while using `Document.normalizeDocument()`.

Since:

DOM Level 3

---

getWholeText

`java.lang.String` **`getWholeText`**`()`

Returns all text of `Text` nodes logically-adjacent text nodes to this node, concatenated in document order.
For instance, in the example below `wholeText` on the `Text` node that contains "bar" returns "barfoo", while on the `Text` node that contains "foo" it returns "barfoo".

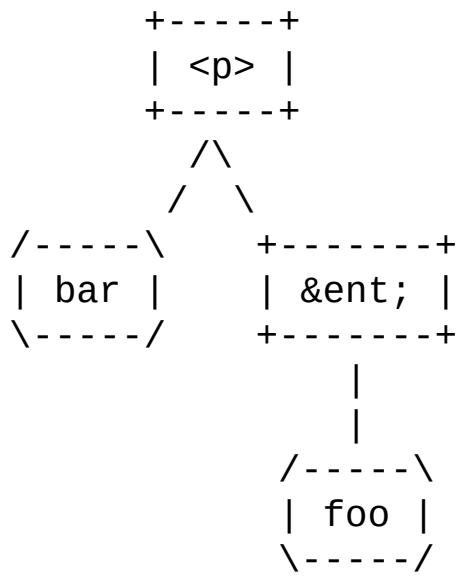


Figure: barTextNode.wholeText value is "barfoo"

Since:

DOM Level 3

---

so, this method will remove all the receiving nodes then insert ONLY ONE text node into this tree.

for example, if want to merge all the text node into one, simply get the whole data then set into the same data as one node (super idea)

replaceWholeText

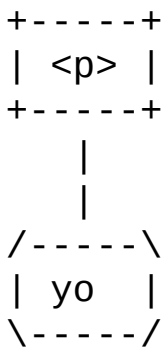
`Text` **`replaceWholeText`**`(ja`
`throws DOMException`

Replaces the text of the current node and all logically-adjacent text nodes with the specified text. All logically-adjacent text nodes are removed including the current node unless it was the recipient of the replacement text.

This method returns the node which received the replacement text. The returned node is:

- `null`, when the replacement text is the empty string;
- the current node, except when the current node is read-only;
- a new `Text` node of the same type (`Text` or `CDATASection`) as the current node inserted at the location of the replacement.

For instance, in the above example calling `replaceWholeText` on the `Text` node that contains "bar" with "yo" in argument results in the following:



many node become into single node

Figure: barTextNode.replaceWholeText("yo") modifies the textual content of barTextNode with "yo"

~~Where the nodes to be removed are read-only descendants of an `EntityReference`, the `EntityReference` must be removed instead of the read-only nodes. If any `EntityReference` to be removed has descendants that are not `EntityReference`, `Text`, or `CDATASection` nodes, the `replaceWholeText` method must fail before performing any modification of the document, raising a `DOMException` with the code `NO_MODIFICATION_ALLOWED_ERR`.~~

~~For instance, in the example below calling `replaceWholeText` on the `Text` node that~~

contains "bar" fails, because the `EntityReference` node "ent" contains an `Element` node which cannot be removed.

Parameters:

`content` - The content of the replacing `Text` node.

Returns:

The `Text` node created with the specified content.

Throws:

`DOMException` - NO_MODIFICATION_ALLOWED_ERR: Raised if one of the `Text` nodes being replaced is readonly.

Since:

DOM Level 3

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

21 - Oct - 08

19 - Dec - 08

org.w3c.dom.traversal

Interface TreeWalker

Tree walker can be ONLY used with SubTree. NodeIterator can be subtree / nodelist /xpath query result

12 methods

3 - getter of constructor parameters
2 - Child Node
2 - Sibling Node
2 - getter / setter for current Node
3 - next / previous / parent Node getter

```
public interface TreeWalker
```

`TreeWalker` objects are used to navigate a document tree or subtree using the view of the document defined by their `whatToShow` flags and filter (if any). Any function which performs navigation using a `TreeWalker` will automatically support any view defined by a `TreeWalker`.

Omitting nodes from the logical view of a subtree can result in a structure that is substantially different from the same subtree in the complete, unfiltered document. Nodes that are siblings in the `TreeWalker` view may be children of different, widely separated nodes in the original view. For instance, consider a `NodeFilter` that skips all nodes except for Text nodes and the root node of a document. In the logical view that results, all text nodes will be siblings and appear as direct children of the root node, no matter how deeply nested the structure of the original document.

See also the Document Object Model (DOM) Level 2 Traversal and Range Specification.

Since:

DOM Level 2

Method Summary

Node Child	**`firstChild`**`()` Moves the `TreeWalker` to the first visible child of the current node, and returns the new node.
Node Current	**`getCurrentNode`**`()` The node at which the `TreeWalker` is currently positioned.

boolean	**getExpandEntityReferences**() The value of this flag determines whether the children of entity reference nodes are visible to the TreeWalker.
NodeFilter	**getFilter**() The filter used to screen nodes.
Node	**getRoot**() The root node of the TreeWalker, as specified when it was created.
int	**getWhatToShow**() This attribute determines which node types are presented via the TreeWalker.
Node Child	**lastChild**() Moves the TreeWalker to the last visible child of the current node, and returns the new node.
Node	**nextNode**() Moves the TreeWalker to the next visible node in document order relative to the current node, and returns the new node.
Node Sibling	**nextSibling**() Moves the TreeWalker to the next sibling of the current node, and returns the new node.
Node	**parentNode**() Moves to and returns the closest visible ancestor node of the current node.
Node	**previousNode**() Moves the TreeWalker to the previous visible node in document order relative to the current node, and returns the new node.
Node Sibling	**previousSibling**() Moves the TreeWalker to the previous sibling of the current node, and returns the new node.
void Current	**setCurrentNode**(Node currentNode) The node at which the TreeWalker is currently positioned.

Method Detail

getRoot

Node **getRoot**()

The root node of the TreeWalker, as specified when it was created.

---

getWhatToShow

int **getWhatToShow**()

This attribute determines which node types are presented via the TreeWalker. The available set of constants is defined in the NodeFilter interface. Nodes not accepted by whatToShow will be skipped, but their children may still be considered. Note that this skip takes precedence over the filter, if any.

---

getFilter

NodeFilter **getFilter**()

The filter used to screen nodes.

---

getExpandEntityReferences

boolean **getExpandEntityReferences**()

The value of this flag determines whether the children of entity reference nodes are visible to the TreeWalker. If false, these children and their descendants will be rejected. Note that this rejection takes precedence over whatToShow and the filter, if any.
To produce a view of the document that has entity references expanded and does not expose the entity reference node itself, use the whatToShow flags to hide the entity reference node and set expandEntityReferences to true when creating the TreeWalker. To produce a view of the document that has entity reference nodes but no entity expansion, use the whatToShow flags to show the entity reference node and set expandEntityReferences to false.

---

getCurrentNode

`Node` **`getCurrentNode`**`()`

The node at which the `TreeWalker` is currently positioned.
Alterations to the DOM tree may cause the current node to no longer be accepted by the `TreeWalker`'s associated filter. `currentNode` may also be explicitly set to any node, whether or not it is within the subtree specified by the `root` node or would be accepted by the filter and `whatToShow` flags. Further traversal occurs relative to `currentNode` even if it is not part of the current view, by applying the filters in the requested direction; if no traversal is possible, `currentNode` is not changed.

---

setCurrentNode

```
void setCurrentNode(Node currentNode)
                    throws DOMException
```

The node at which the `TreeWalker` is currently positioned.
Alterations to the DOM tree may cause the current node to no longer be accepted by the `TreeWalker`'s associated filter. `currentNode` may also be explicitly set to any node, whether or not it is within the subtree specified by the `root` node or would be accepted by the filter and `whatToShow` flags. Further traversal occurs relative to `currentNode` even if it is not part of the current view, by applying the filters in the requested direction; if no traversal is possible, `currentNode` is not changed.

Throws:

`DOMException` - NOT_SUPPORTED_ERR: Raised if an attempt is made to set `currentNode` to `null`.

---

parentNode

`Node` **`parentNode`**`()`

Moves to and returns the closest visible ancestor node of the current node. If the search for `parentNode` attempts to step upward from the `TreeWalker`'s `root` node, or if it fails to find a visible ancestor node, this method retains the current position and returns `null`.

Returns:

The new parent node, or `null` if the current node has no parent in the `TreeWalker`'s logical view.

---

firstChild

`Node` **`firstChild()`**

Moves the `TreeWalker` to the first visible child of the current node, and returns the new node. If the current node has no visible children, returns `null`, and retains the current node.

Returns:

The new node, or `null` if the current node has no visible children in the `TreeWalker`'s logical view.

---

lastChild

`Node` **`lastChild()`**

Moves the `TreeWalker` to the last visible child of the current node, and returns the new node. If the current node has no visible children, returns `null`, and retains the current node.

Returns:

The new node, or `null` if the current node has no children in the `TreeWalker`'s logical view.

---

previousSibling

`Node` **`previousSibling()`**

Moves the `TreeWalker` to the previous sibling of the current node, and returns the new node. If the current node has no visible previous sibling, returns `null`, and retains the current node.

Returns:

The new node, or `null` if the current node has no previous sibling. in the `TreeWalker`'s logical view.

---

nextSibling

`Node` **`nextSibling()`**

Moves the `TreeWalker` to the next sibling of the current node, and returns the new node. If the current node has no visible next sibling, returns `null`, and retains the current node.

Returns:

The new node, or `null` if the current node has no next sibling. in the `TreeWalker`'s logical view.

---

previousNode

`Node` **`previousNode()`**

Moves the `TreeWalker` to the previous visible node in document order relative to the current node, and returns the new node. If the current node has no previous node, or if the search for `previousNode` attempts to step upward from the `TreeWalker`'s `root` node, returns `null`, and retains the current node.

Returns:

The new node, or `null` if the current node has no previous node in the `TreeWalker`'s logical view.

---

nextNode

Node nextNode()

Moves the `TreeWalker` to the next visible node in document order relative to the current node, and returns the new node. If the current node has no next node, or if the search for nextNode attempts to step upward from the `TreeWalker`'s `root` node, returns `null`, and retains the current node.

Returns:

The new node, or `null` if the current node has no next node in the `TreeWalker`'s logical view.

---

18 - Dec - 08

it has 3 methods

org.w3c.dom

Interface TypeInfo

`public interface` **`TypeInfo`**

The `TypeInfo` interface represents a type referenced from `Element` or `Attr` nodes, specified in the schemas associated with the document. The type is a pair of a namespace URI and name properties, and depends on the document's schema.

~~If the document's schema is an XML DTD [XML 1.0], the values are computed as follows:~~

- ~~If this type is referenced from an `Attr` node, `typeNamespace` is `"http://www.w3.org/TR/REC-xml"` and `typeName` represents the **[attribute type]** property in the [XML Information Set]. If there is no declaration for the attribute, `typeNamespace` and `typeName` are `null`~~
- ~~If this type is referenced from an `Element` node, `typeNamespace` and `typeName` are `null`.~~

~~If the document's schema is an XML Schema [XML Schema Part 1], the values are computed as follows using the post-schema-validation infoset contributions (also called PSVI contributions):~~

- ~~If the **[validity]** property exists AND is *"invalid"* or *"notKnown"*: the {target namespace} and {name} properties of the declared type if available, otherwise `null`.~~

  ~~**Note:** At the time of writing, the XML Schema specification does not require exposing the declared type. Thus, DOM implementations might choose not to provide type information if validity is not valid.~~
- ~~If the **[validity]** property exists and is *"valid"*:~~
  1. ~~If **[member type definition]** exists:~~
     1. ~~If {name} is not absent, then expose {name} and {target namespace} properties of the **[member type definition]** property;~~
     2. ~~Otherwise, expose the namespace and local name of the corresponding anonymous type name.~~

2. If the **[type definition]** property exists:
    1. If {name} is not absent, then expose {name} and {target namespace} properties of the **[type definition]** property;
    2. Otherwise, expose the namespace and local name of the corresponding anonymous type name.
3. If the **[member type definition anonymous]** exists:
    1. If it is false, then expose **[member type definition name]** and **[member type definition namespace]** properties;
    2. Otherwise, expose the namespace and local name of the corresponding anonymous type name.
4. If the **[type definition anonymous]** exists:
    1. If it is false, then expose **[type definition name]** and **[type definition namespace]** properties;
    2. Otherwise, expose the namespace and local name of the corresponding anonymous type name.

Note: Other schema languages are outside the scope of the W3C and therefore should define how to represent their type systems using `TypeInfo`.

See also the Document Object Model (DOM) Level 3 Core Specification.

Since:

DOM Level 3

Field Summary

`static int`	**`DERIVATION_EXTENSION`** If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the derivation by extension.
`static int`	**`DERIVATION_LIST`** If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the list.
`static int`	**`DERIVATION_RESTRICTION`** If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the derivation by restriction if complex types are involved, or a restriction if simple types are involved.

static int	**DERIVATION_UNION** If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the union if simple types are involved.

Method Summary

java.lang.String	**getTypeName**() The name of a type declared for the associated element or attribute, or null if unknown.
java.lang.String	**getTypeNamespace**() The namespace of the type declared for the associated element or attribute or null if the element does not have declaration or if no namespace information is available.
boolean	**isDerivedFrom**(java.lang.String typeNamespaceArg, java.lang.String typeNameArg, int derivationMethod) This method returns if there is a derivation between the reference type definition, i. e. the TypeInfo on which the method is being called, and the other type definition, i.e. the one passed as parameters.

Field Detail

DERIVATION_RESTRICTION

static final int **DERIVATION_RESTRICTION**

If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the derivation by restriction if complex types are involved, or a restriction if simple types are involved.
The reference type definition is derived by restriction from the other type definition if the other type definition is the same as the reference type definition, or if the other type definition can be reached recursively following the {base type definition} property from the reference type definition, and all the *derivation methods* involved are restriction.

See Also:

Constant Field Values

DERIVATION_EXTENSION

```
static final int DERIVATION_EXTENSION
```

If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the derivation by extension.
The reference type definition is derived by extension from the other type definition if the other type definition can be reached recursively following the {base type definition} property from the reference type definition, and at least one of the *derivation methods* involved is an extension.

See Also:

Constant Field Values

DERIVATION_UNION

```
static final int DERIVATION_UNION
```

If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the union if simple types are involved.
The reference type definition is derived by union from the other type definition if there exists two type definitions T1 and T2 such as the reference type definition is derived from T1 by `DERIVATION_RESTRICTION` or `DERIVATION_EXTENSION`, T2 is derived from the other type definition by `DERIVATION_RESTRICTION`, T1 has {variety} *union*, and one of the {member type definitions} is T2. Note that T1 could be the same as the reference type definition, and T2 could be the same as the other type definition.

See Also:

Constant Field Values

DERIVATION_LIST

```
static final int DERIVATION_LIST
```

If the document's schema is an XML Schema [XML Schema Part 1] , this constant represents the list.

The reference type definition is derived by list from the other type definition if there exists two type definitions T1 and T2 such as the reference type definition is derived from T1 by `DERIVATION_RESTRICTION` or `DERIVATION_EXTENSION`, T2 is derived from the other type definition by `DERIVATION_RESTRICTION`, T1 has {variety} *list*, and T2 is the {item type definition}. Note that T1 could be the same as the reference type definition, and T2 could be the same as the other type definition.

See Also:

Constant Field Values

Method Detail

getTypeName

```
java.lang.String getTypeName()
```

The name of a type declared for the associated element or attribute, or `null` if unknown.

---

getTypeNamespace

```
java.lang.String getTypeNamespace()
```

The namespace of the type declared for the associated element or attribute or `null` if the element does not have declaration or if no namespace information is available.

---

isDerivedFrom

```
boolean isDerivedFrom(java.lang.String typeNamespaceArg,
                      java.lang.String typeNameArg,
                      int derivationMethod)
```

~~This method returns if there is a derivation between the reference type definition, i.e. the `TypeInfo` on which the method is being called, and the other type definition, i.e. the one passed as parameters.~~