Web DOM Core

Work in Progress — Last Update 18 August 2010

Editors

Simon Pieters <simonp@opera.com>
Geoffrey Sneddon <gsneddon@opera.com>
Ms2ger <ms2ger@gmail.com>

PDF print version

Letter

Version history

http://bitbucket.org/ms2ger/web-dom-core
http://hg.gsnedders.com/web-dom-core
http://simon.html5.org/specs/web-dom-core

Issues

**
- innerHTML etc on all Elements / `Document`[p16] `implements HTMLDocument`[HTML]; ? public-html/2007Aug/0778.html, public-webapi/2007Aug/0069.html, public-webapi/2007Aug/0070.html
- \0 http://krijnhoetmer.nl/irc-logs/whatwg/20080321#l-312
- document.parseError? http://www.w3.org/mid/op.ucv5axjp64w2qv@annevk-t60.oslo.opera.com
- sourceIndex? http://www.quirksmode.org/dom/w3c_core.html
- [Reflect] http://krijnhoetmer.nl/irc-logs/whatwg/20090622#l-90
- Perhaps we should move DOMStringMap to this spec...

**
- http://hg.mozilla.org/mozilla-central/rev/91694d19d7b2

**

Abstract

This specification defines the DOM Core part of the Web platform. The Document Object Model is a language- and platform neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. Web DOM Core mostly subsets DOM3 Core, but redefines some things and adds some features that were widely implemented already.

Table of contents

1 Common infrastructure

1.1 Terminology

The term **tree order** means a pre-order, depth-first traversal of DOM nodes involved (through the `parentNode`p13/`childNodes`p13 relationship).

The term **context node** means the `Node`p12 on which the method or attribute being discussed was called.

The term **root element**, when not explicitly qualified as referring to the document's root element, means the furthest ancestor element node of whatever node is being discussed, or the node itself if it has no ancestors. When the node is a part of the document, then the node's root elementp5 is indeed the document's root element; however, if the node is not currently part of the document tree, the root element will be an orphaned node.

When an element's root elementp5 is the root element of a `Document`p16, it is said to be **in a `Document`**.

A node's **home subtree** is the subtree rooted at that node's root elementp5. When a node is in a `Document`p5, its home subtreep5 is that `Document`p16's tree.

The **space characters**, for the purposes of this specification, are U+0020 SPACE, U+0009 CHARACTER TABULATION (tab), U+000A LINE FEED (LF), U+000C FORM FEED (FF), and U+000D CARRIAGE RETURN (CR).

1.2 Conformance requirements

All diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC2119. For readability, these words do not appear in all uppercase letters in this specification. [RFC2119]p33

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps may be implemented in any manner, so long as the end result is equivalent. (In particular, the algorithms defined in this specification are intended to be easy to follow, and not intended to be performant.)

User agents may impose implementation-specific limits on otherwise unconstrained inputs, e.g. to prevent denial of service attacks, to guard against running out of memory, or to work around platform-specific limitations.

When a method or an attribute is said to call another method or attribute, the user agent must invoke its internal API for that attribute or method so that e.g. the author can't change the behavior by overriding attributes or methods with custom properties or functions in ECMAScript.

Unless otherwise stated, string comparisons are done in a case-sensitivep6 manner.

1.2.1 Dependencies

The IDL fragments in this specification must be interpreted as required for conforming IDL fragments, as described in the Web IDL specification. [WEBIDL]p33

Except where otherwise specified, if an IDL attribute that is a floating point number type (`float`[WEBIDL]) is assigned an Infinity or Not-a-Number (NaN) value, a `NOT_SUPPORTED_ERR`p10 exception must be raised.

Except where otherwise specified, if a method with an argument that is a floating point number type (`float`[WEBIDL]) is passed an Infinity or Not-a-Number (NaN) value, a `NOT_SUPPORTED_ERR`p10 exception must be raised.

Some of the terms used in this specification are defined in *Web IDL*, *XML*, *Namespaces in XML* and *HTML*. [WEBIDL]p33 [XML]p33 [XMLNS]p33 [HTML]p33

1.2.2 Extensibility

Vendor-specific proprietary extensions to this specification are strongly discouraged. Authors must not use such extensions, as doing so reduces interoperability and fragments the user base, allowing only users of specific user agents to access the content in question.

If vendor-specific extensions are needed, the members should be prefixed by vendor-specific strings to prevent clashes with future versions of this specification. Extensions must be defined so that the use of extensions neither contradicts nor causes the non-conformance of functionality defined in the specification.

When vendor-neutral extensions to this specification are needed, either this specification can be updated accordingly, or an extension specification can be written that overrides the requirements in this specification. When someone applying this specification to their activities decides that they will recognise the requirements of such an extension specification, it becomes an **applicable specification** for the purposes of conformance requirements in this specification.

1.3 Case-sensitivity

This specification defines several comparison operators for strings.

Comparing two strings in a **case-sensitive** manner means comparing them exactly, codepoint for codepoint.

Comparing two strings in a **ASCII case-insensitive** manner means comparing them exactly, codepoint for codepoint, except that the characters in the range U+0041 .. U+005A (i.e. LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) and the corresponding characters in the range U+0061 .. U+007A (i.e. LATIN SMALL LETTER A to LATIN SMALL LETTER Z) are considered to also match.

Converting a string to uppercase means replacing all characters in the range U+0061 .. U+007A (i.e. LATIN SMALL LETTER A to LATIN SMALL LETTER Z) with the corresponding characters in the range U+0041 .. U+005A (i.e. LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z).

Converting a string to lowercase means replacing all characters in the range U+0041 .. U+005A (i.e. LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) with the corresponding characters in the range U+0061 .. U+007A (i.e. LATIN SMALL LETTER A to LATIN SMALL LETTER Z).

1.4 Common microsyntaxes

1.4.1 Common parser idioms

Some of the micro-parsers described below follow the pattern of having an *input* variable that holds the string being parsed, and having a *position* variable pointing at the next character to parse in *input*.

For parsers based on this pattern, a step that requires the user agent to **collect a sequence of characters** means that the following algorithm must be run, with *characters* being the set of characters that can be collected:

1. Let *input* and *position* be the same variables as those of the same name in the algorithm that invoked these steps.
2. Let *result* be the empty string.
3. While *position* doesn't point past the end of *input* and the character at *position* is one of the *characters*, append that character to the end of *result* and advance *position* to the next character in *input*.
4. Return *result*.

The step **skip whitespace** means that the user agent must collect a sequence of characters[p6] that are space characters[p5]. The collected characters are not used.

1.4.2 Space-separated tokens

A **set of space-separated tokens** is a string containing zero or more words separated by one or more space characters[p5], where words consist of any string of one or more characters, none of which are space characters[p5].

A string containing a set of space-separated tokens[p6] may have leading or trailing space characters[p5].

When a user agent has to **split a string on spaces**, it must use the following algorithm:

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.

3. Let *tokens* be a list of tokens, initially empty.
4. Skip whitespace[p6]
5. While *position* is not past the end of *input*:
    1. Collect a sequence of characters[p6] that are not space characters[p5].
    2. Add the string collected in the previous step to *tokens*.
    3. Skip whitespace[p6]
6. Return *tokens*.

When a user agent has to **remove a token from a string**, it must use the following algorithm:

1. Let *input* be the string being modified.
2. Let *token* be the token being removed. It will not contain any space characters[p5].
3. Let *output* be the output string, initially empty.
4. Let *position* be a pointer into *input*, initially pointing at the start of the string.
5. *Loop*: If *position* is beyond the end of *input*, abort these steps.
6. If the character at *position* is a space character[p5]:
    1. Append the character at *position* to the end of *output*.
    2. Advance *position* so it points at the next character in *input*.
    3. Return to the step labeled *loop*.
7. Otherwise, the character at *position* is the first character of a token. Collect a sequence of characters[p6] that are not space characters[p5], and let that be *s*.
8. If *s* is exactly equal to *token*, then:
    1. Skip whitespace[p6] (in *input*).
    2. Remove any space characters[p5] currently at the end of *output*.
    3. If *position* is not past the end of *input*, and *output* is not the empty string, append a single U+0020 SPACE character at the end of *output*.
9. Otherwise, append *s* to the end of *output*.
10. Return to the step labeled *loop*.

Note: This causes any occurrences of the token to be removed from the string, and any spaces that were surrounding the token to be collapsed to a single space, except at the start and end of the string, where such spaces are removed.

1.5 DOM features

A **DOM feature** is a unique, ASCII case-insensitive[p6] string that represents a certain feature of the user agent.

A **DOM feature version** is a (*feature string*, *version*) tuple, where *feature string* is DOM feature[p7] and *version* is a case-sensitive[p6] string representing a version number.

Specifications may define which DOM features[p7] a user agent is to **support**, as well as an associated list of one or more case-sensitive[p6] strings representing version numbers, and under which circumstances.

A user agent must **support** a DOM feature version[p7] (*feature string*, *version*) if it supports[p7] a DOM feature[p7] that is a ASCII case-insensitive[p6] match for *feature string* and *version* is in the associated list of versions.

A user agent must support[p7] the (*feature*, "") tuple if it supports[p7] a DOM feature[p7] that is a ASCII case-insensitive[p6] match for *feature string*.

Note: Authors are strongly discouraged from using DOM features[p7], as they are notoriously unreliable and imprecise. Authors are encouraged to rely on explicit feature testing or graceful degradation.

For historical reasons, user agents must support[p7] the "XML" DOM feature[p7] with the versions "1.0" and "2.0" associated with it, and the "Core" DOM feature[p7] with the version "2.0" associated with it.

1.6 Cloning nodes

When a UA is to **clone** a *node*, with a *new ownerDocument* and with a *clone children* flag, it must run the following steps:

1. If *node* is a `DocumentType`[p24] node, raise a `NOT_SUPPORTED_ERR`[p10] exception and abort these steps.
2. Let *copy* be a new `Node`[p12] that implements the same interfaces as *node*, with `ownerDocument`[p13] set to *new ownerDocument*, `prefix`[p15], `localName`[p15] and `namespaceURI`[p15] attributes set to the values of the attributes on *node* with the same names, and other attributes set to the values of the attributes on *node* with the same names depending on the type of *node* according to the following table:

Type of *node*	Attributes
`Element`[p22]	—
`Attr`[p21]	`value`[p21]
`Text`[p25]	`data`[p24]
`ProcessingInstruction`[p24]	`target`[p24], `data`[p24]
`Comment`[p25]	`data`[p24]
`DocumentFragment`[p16]	—

** 3. If *node* is an `Element`[p22] node, copy its attributes.
4. If the *clone children* flag is set, clone[p8] all the children of *node* and append them to *copy*, with the same *new ownerDocument* and the *clone children* flag being set.
5. Return *copy*.

1.7 Legal hierarchy

A `Node`[p12] is said to have a **legal hierarchy** if all the following conditions are true:

- The `Node`[p12] is a `Document`[p16] node or an `Attr`[p21] node and has no parent node.
- The `Node`[p12] is a `Document`[p16] node and has no child `Text`[p25] nodes.
- The `Node`[p12] is a `Document`[p16] node and has no more than one child `Element`[p22] node.
- The `Node`[p12] is an `Attr`[p21] node, a `Text`[p25] node, a `ProcessingInstruction`[p24] node, a `Comment`[p25] node, or a `DocumentType`[p24] node, and has no child nodes.

Before running the steps of an algorithm of a method or attribute in this specification, the user agent must check that running the algorithm will result in a legal hierarchy[p8]. If it won't, then the user agent must instead raise a `HIERARCHY_REQUEST_ERR`[p10] exception.

1.8 Namespaces

The **HTML namespace** is `http://www.w3.org/1999/xhtml`.

The **XML namespace** is `http://www.w3.org/XML/1998/namespace`.

The **XMLNS namespace** is `http://www.w3.org/2000/xmlns/`.

2 Basic types

A **DOMTimeStamp** represents a number of milliseconds.

```
typedef unsigned long long DOMTimeStamp;
```

3 Exceptions

3.1 Exception *DOMException*[p10]

```
exception DOMException {
  const unsigned short INDEX_SIZE_ERR = 1;
  const unsigned short DOMSTRING_SIZE_ERR = 2; // historical
  const unsigned short HIERARCHY_REQUEST_ERR = 3;
  const unsigned short WRONG_DOCUMENT_ERR = 4;
  const unsigned short INVALID_CHARACTER_ERR = 5;
  const unsigned short NO_DATA_ALLOWED_ERR = 6;
  const unsigned short NO_MODIFICATION_ALLOWED_ERR = 7;
  const unsigned short NOT_FOUND_ERR = 8;
  const unsigned short NOT_SUPPORTED_ERR = 9;
  const unsigned short INUSE_ATTRIBUTE_ERR = 10;
  const unsigned short INVALID_STATE_ERR = 11;
  const unsigned short SYNTAX_ERR = 12;
  const unsigned short INVALID_MODIFICATION_ERR = 13;
  const unsigned short NAMESPACE_ERR = 14;
  const unsigned short INVALID_ACCESS_ERR = 15;
  const unsigned short VALIDATION_ERR = 16;
  const unsigned short TYPE_MISMATCH_ERR = 17;
  const unsigned short SECURITY_ERR = 18;
  const unsigned short NETWORK_ERR = 19;
  const unsigned short ABORT_ERR = 20;
  const unsigned short URL_MISMATCH_ERR = 21;
  const unsigned short QUOTA_EXCEEDED_ERR = 22;
  const unsigned short TIMEOUT_ERR = 23;
  const unsigned short PARSE_ERR = 81;
  const unsigned short SERIALIZE_ERR = 82;
  unsigned short code;
  DOMString message;
  DOMString name;
};
```

The `code` exception member must return the code for the exception, which must be one of the following:

1. **`INDEX_SIZE_ERR`**
2. **`DOMSTRING_SIZE_ERR`**
3. **`HIERARCHY_REQUEST_ERR`**
4. **`WRONG_DOCUMENT_ERR`**
5. **`INVALID_CHARACTER_ERR`**
6. **`NO_DATA_ALLOWED_ERR`**
7. **`NO_MODIFICATION_ALLOWED_ERR`**
8. **`NOT_FOUND_ERR`**
9. **`NOT_SUPPORTED_ERR`**
10. **`INUSE_ATTRIBUTE_ERR`**
11. **`INVALID_STATE_ERR`**
12. **`SYNTAX_ERR`**
13. **`INVALID_MODIFICATION_ERR`**
14. **`NAMESPACE_ERR`**
15. **`INVALID_ACCESS_ERR`**
16. **`VALIDATION_ERR`**
17. **`TYPE_MISMATCH_ERR`**
18. **`SECURITY_ERR`**
19. **`NETWORK_ERR`**
20. **`ABORT_ERR`**
21. **`URL_MISMATCH_ERR`**
22. **`QUOTA_EXCEEDED_ERR`**
23. **`TIMEOUT_ERR`**
81. **`PARSE_ERR`**
82. **`SERIALIZE_ERR`**

** Add a description of those exceptions?

The `message` exception member must return a User Agent-defined human readable string describing the exception.

The **`name`** exception member must return the name of the exception constant as a string.

4 Nodes

4.1 Interface *Node*[p12]

```
interface Node {

  // NodeType
  const unsigned short ELEMENT_NODE = 1;
  const unsigned short ATTRIBUTE_NODE = 2;
  const unsigned short TEXT_NODE = 3;
  const unsigned short CDATA_SECTION_NODE = 4; // historical
  const unsigned short ENTITY_REFERENCE_NODE = 5; // historical
  const unsigned short ENTITY_NODE = 6; // historical
  const unsigned short PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short COMMENT_NODE = 8;
  const unsigned short DOCUMENT_NODE = 9;
  const unsigned short DOCUMENT_TYPE_NODE = 10;
  const unsigned short DOCUMENT_FRAGMENT_NODE = 11;
  const unsigned short NOTATION_NODE = 12; // historical

  readonly attribute DOMString nodeName;
           attribute DOMString nodeValue;
  readonly attribute unsigned short nodeType;
  readonly attribute Node parentNode;
  readonly attribute Element parentElement;
  readonly attribute NodeList childNodes;
  readonly attribute Node firstChild;
  readonly attribute Node lastChild;
  readonly attribute Node previousSibling;
  readonly attribute Node nextSibling;
  readonly attribute NamedNodeMap attributes;
  readonly attribute Document ownerDocument;
  Node insertBefore(in Node newChild, in Node refChild);
  Node replaceChild(in Node newChild, in Node oldChild);
  Node removeChild(in Node oldChild);
  Node appendChild(in Node newChild);
  boolean hasChildNodes();
  Node cloneNode(in boolean deep);
  boolean isSupported([TreatNullAs=EmptyString] in DOMString feature, in DOMString version);
  readonly attribute DOMString namespaceURI;
  readonly attribute DOMString prefix;
  readonly attribute DOMString localName;
  boolean hasAttributes();
  readonly attribute DOMString baseURI;

  // DocumentPosition
  const unsigned short DOCUMENT_POSITION_DISCONNECTED = 0x01;
  const unsigned short DOCUMENT_POSITION_PRECEDING = 0x02;
  const unsigned short DOCUMENT_POSITION_FOLLOWING = 0x04;
  const unsigned short DOCUMENT_POSITION_CONTAINS = 0x08;
  const unsigned short DOCUMENT_POSITION_CONTAINED_BY = 0x10;
  const unsigned short DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC = 0x20;

  unsigned short compareDocumentPosition(in Node other);

  [TreatNullAs=EmptyString] attribute DOMString textContent;
  boolean isSameNode(in Node other);
  DOMString lookupPrefix(in DOMString namespaceURI);
  boolean isDefaultNamespace(in DOMString namespaceURI);
  DOMString lookupNamespaceURI(in DOMString prefix);
  boolean isEqualNode(in Node arg);
};
```

1. **`ELEMENT_NODE`**

2. **`ATTRIBUTE_NODE`**
3. **`TEXT_NODE`**
4. **`CDATA_SECTION_NODE`**
5. **`ENTITY_REFERENCE_NODE`**
6. **`ENTITY_NODE`**
7. **`PROCESSING_INSTRUCTION_NODE`**
8. **`COMMENT_NODE`**
9. **`DOCUMENT_NODE`**
10. **`DOCUMENT_TYPE_NODE`**
11. **`DOCUMENT_FRAGMENT_NODE`**
12. **`NOTATION_NODE`**

The **`nodeName`**, **`nodeValue`** and **`nodeType`** attributes must, on getting, return what is in the second, third and forth column, respectively, if the node also implements the interface in the first column on the same row in the following table:

Interface	`nodeName` p13	`nodeValue` p13	`nodeType` p13
`Element` p22	same as `tagName` p22	null	1
`Attr` p21	same as `name` p21	same as `value` p21	2
`Text` p25	"`#text`"	same as `data` p24	3
`ProcessingInstruction` p24	same as `target` p24	same as `data` p24	7
`Comment` p25	"`#comment`"	same as `data` p24	8
`Document` p16	"`#document`"	null	9
`DocumentType` p24	same as `name` p24	null	10
`DocumentFragment` p16	"`#document-fragment`"	null	11

The **`parentNode`** attribute must, on getting, run the following steps:

1. If the context node p5 is an `Attr` p21 node, return null and abort these steps.
2. If the context node p5 doesn't have a parent node, return null and abort these steps.
3. Return the parent node of the context node p5.

The **`parentElement`** attribute must, on getting, return the parent node of the context node p5 if there is a parent and it is an element, or null otherwise.

The **`childNodes`** attribute must, on getting, return a `NodeList` p26 rooted at the context node p5 matching only child nodes.

The **`firstChild`** attribute must, on getting, return the first child node of the context node p5, or null if there is none.

The **`lastChild`** attribute must, on getting, return the last child node of the context node p5, or null if there is none.

The **`previousSibling`** attribute must, on getting, run the following steps:

1. If the context node p5 is an `Attr` p21 node, return null and abort these steps.
2. If the context node p5 doesn't have a previous sibling node, return null and abort these steps.
3. Return the previous sibling node of the context node p5.

The **`nextSibling`** attribute must, on getting, run the following steps:

1. If the context node p5 is an `Attr` p21 node, return null and abort these steps.
2. If the context node p5 doesn't have a next sibling node, return null and abort these steps.
3. Return the next sibling node of the context node p5.

The **`attributes`** attribute must, on getting, return a `NamedNodeMap` p27 of all the `Attr` p21 nodes associated with the node of the context node p5, if it is an `Element` p22 node, or null otherwise.

The **`ownerDocument`** attribute must, on getting, return the `Document` p16 node that the context node p5 is associated with, or null if there is none.

The **`insertBefore(`*`newChild`*`, `*`refChild`*`)`** method must run the following steps:

1. If the context node[p5] is an `Attr`[p21] node or a `Text`[p25] node, then raise a `HIERARCHY_REQUEST_ERR`[p10] and abort these steps.
2. If *newChild* is null, then raise a `NOT_SUPPORTED_ERR`[p10] exception and abort these steps.
3. If *refChild* is not null and is not a child of the context node[p5], then raise a `NOT_FOUND_ERR`[p10] exception and abort these steps.
4. If *newChild*'s `ownerDocument`[p13] is not equal to the context node[p5]'s `ownerDocument`[p13], call the context node[p5]'s `ownerDocument`[p13] `adoptNode`[p19] method with *newChild* as its argument.
5. If *newChild* is a `DocumentFragment`[p16] node, then while *newChild*'s `firstChild`[p13] is not null, call `insertBefore`[p13] on the context node[p5] with *newChild*'s `firstChild`[p13] as first argument and *refChild* as second argument.
6. Otherwise, if *refChild* is null, append *newChild* to the context node[p5].
7. Otherwise insert *newChild* in the context node[p5] as the previous sibling of *refChild*.
8. Return *newChild*.

The **`replaceChild(`*`newChild, oldChild`*`)`** method must run the following steps:

1. If the context node[p5] is an `Attr`[p21] node or a `Text`[p25] node, then raise a `HIERARCHY_REQUEST_ERR`[p10] and abort these steps.
2. If either *newChild* or *oldChild* is null, then raise a `NOT_SUPPORTED_ERR`[p10] exception and abort these steps.
3. If *newChild*'s `ownerDocument`[p13] is not equal to the context node[p5]'s `ownerDocument`[p13], call the context node[p5]'s `ownerDocument`[p13] `adoptNode`[p19] method with *newChild* as its argument.
4. If *oldChild* is not a child of the context node[p5], then raise a `NOT_FOUND_ERR`[p10] exception and abort these steps.
5. Let *refChild* be *oldChild*'s `nextSibling`[p13].
6. Remove *oldChild* from context node[p5].
7. Call `insertBefore`[p13] on the context node[p5] with *newChild* and *refChild* as arguments, respectively.
8. Return *newChild*.

The **`removeChild(`*`oldChild`*`)`** method must run the following steps:

1. If *oldChild* is null, then raise a `NOT_SUPPORTED_ERR`[p10] exception and abort these steps.
2. If *oldChild* is not a child of the context node[p5], then raise a `NOT_FOUND_ERR`[p10] exception and abort these steps.
3. Remove *oldChild* from context node[p5].
4. Return *oldChild*.

The **`appendChild(`*`newChild`*`)`** method must run the following steps:

1. If the context node[p5] is an `Attr`[p21] node or a `Text`[p25] node, then raise a `HIERARCHY_REQUEST_ERR`[p10] and abort these steps.
2. If *newChild* is null, then raise a `NOT_SUPPORTED_ERR`[p10] exception and abort these steps.
3. If *newChild*'s `ownerDocument`[p13] is not equal to the context node[p5]'s `ownerDocument`[p13], call the context node[p5]'s `ownerDocument`[p13] `adoptNode`[p19] method with *newChild* as its argument.
4. Append *newChild* to the context node[p5].
5. Return *newChild*.

The **`hasChildNodes()`** method must return false if the context node[p5]'s `firstChild`[p13] is null, and true otherwise.

The **`cloneNode(`*`deep`*`)`** method must return a clone[p8] of the context node[p5], with *new ownerDocument* being the context node[p5]'s `ownerDocument`[p13], and the *clone children* flag set if *deep* is true.

The **`isSupported(`*`feature, version`*`)`** method must return true if the user agent supports[p7] the (*feature*, *version*) tuple on the context node[p5], and false otherwise.

The **namespaceURI** attribute, on getting, must return the namespace that is associated with the node, if there is one and it's not the empty string, or null otherwise.

The **prefix** attribute, on getting, must return the prefix that is associated with the node, if there is one and it's not the empty string, or

** null otherwise. And on setting?

The **localName** attribute, on getting, must return the local name that is associated with the node, if it has one, and null otherwise.

The **hasAttributes()** method must return whether there are any attributes associated with the context node[p5], if it is an `Element`[p22] node, and false otherwise.

** The **baseURI** attribute must ...

1. **DOCUMENT_POSITION_DISCONNECTED**
2. **DOCUMENT_POSITION_PRECEDING**
4. **DOCUMENT_POSITION_FOLLOWING**
8. **DOCUMENT_POSITION_CONTAINS**
16. **DOCUMENT_POSITION_CONTAINED_BY**
32. **DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC**

** The **compareDocumentPosition(*other*)** method must ...

The **textContent** attribute, on getting, must return a concatenation of the `data`[p24] of all the descendant `Text`[p25] nodes of the context node[p5], in tree order[p5]. On setting, it must run the following steps:

1. Remove all the child nodes of the context node[p5].
2. Let *data* be the value being assigned.
3. If *data* is not the empty string, append a new `Text`[p25] node to the context node[p5] whose `data`[p24] is set to *data*.

** http://www.w3.org/mid/c9e12660808271343v58990698gabac29d2123a82ce@mail.gmail.com

The **isSameNode(*other*)** method must return true if *other* is a reference to the same object as the context node[p5], and false otherwise.

** The **lookupPrefix(*namespaceURI*)** method must ...

** The **isDefaultNamespace(*namespaceURI*)** method must ...

** The **lookupNamespaceURI(*prefix*)** method must ...

** clarify lookupNamespaceURI http://www.w3.org/mid/4878DFC6.40401@lachy.id.au; lookupNamespaceURI, isDefaultNamespace
** http://html5.org/tools/web-apps-tracker?from=2125&to=2126

The **isEqualNode(*arg*)** method must return true if all of the following conditions are true, and must otherwise return false:

- *arg* is not null.
- *arg*'s `nodeType`[p13] is the same as the context node[p5]'s `nodeType`[p13].
- *arg*'s `nodeName`[p13] is the same as the context node[p5]'s `nodeName`[p13].
- *arg*'s `localName`[p15] is the same as the context node[p5]'s `localName`[p15].
- *arg*'s `namespaceURI`[p15] is the same as the context node[p5]'s `namespaceURI`[p15].

- *arg*'s `prefix`p15 is the same as the context nodep5's `prefix`p15.
- *arg*'s `nodeValue`p13 is the same as the context nodep5's `nodeValue`p13.
- Either *arg*'s `attributes`p13 and the context nodep5's `attributes`p13 are both null or a bijection exists between the set of *arg*'s `attributes`p13 and the set of the context nodep5's `attributes`p13 so that every `Attr`p21 node in the former is mapped to an `Attr`p21 node in the latter for which calling `isEqualNode`p15 on the first `Attr`p21 node with the second `Attr`p21 node as its argument returns true.
- *arg*'s `childNodes`p13' `length`p26 is the same as the context nodep5's `childNodes`p13' `length`p26.
- Calling `isEqualNode`p15 on each child node of the context nodep5, with the child node of the same index in *arg* as argument returns true for every child node.

4.2 Interface *DocumentFragment*p16

```
interface DocumentFragment : Node {
};
```

4.3 Interface *Document*p16

```
interface Document : Node {
  readonly attribute DocumentType doctype;
  readonly attribute DOMImplementation implementation;
  readonly attribute Element documentElement;
  readonly attribute WindowProxy[WINDOW] defaultView;

  Element createElement([TreatNullAs=EmptyString] in DOMString tagName);
  Element createElementNS(in DOMString namespaceURI, in DOMString qualifiedName);
  DocumentFragment createDocumentFragment();
  Text createTextNode(in DOMString data);
  Comment createComment(in DOMString data);
  ProcessingInstruction createProcessingInstruction(in DOMString target, in DOMString data);

  NodeList getElementsByTagName(in DOMString tagname);
  NodeList getElementsByTagNameNS(in DOMString namespaceURI, in DOMString localName);
  NodeList getElementsByClassName(in DOMString classNames);
  Element getElementById(in DOMString elementId);

  Node importNode(in Node importedNode, in boolean deep);
  Node adoptNode(in Node source);

  readonly attribute DOMString inputEncoding;
           attribute DOMString documentURI;
  readonly attribute DOMString compatMode;
};
XMLDocument : Document {};
```

A `Document`p16 node is assumed to be an **XML document** unless they are flagged as being an **HTML document** when they are created. Whether a document is an HTML documentp16 or an XML documentp16 affects the behavior of certain APIs.

A `Document`p16 node is always set to one of three modes when it is created: **no-quirks mode**, the default; **quirks mode**, used typically for legacy documents; and **limited-quirks mode**, also known as "almost standards" mode. Unless other applicable specificationsp6 define otherwise, the `Document`p16 must be in no-quirks modep16.

Note: The mode is only ever changed from the default if the `Document`p16 node is created by the HTML parser[HTML], based on the presence, absence, or value of the DOCTYPE string. [HTML]p33

***document*[WINDOW]** **`.defaultView`**p17

Returns the `Window`[WINDOW] object of the active document[HTML].

The **`doctype`** attribute must return the first child of the `Document`p16 node that is a `DocumentType`p24 node, if there is one, or null otherwise.

Note: In both HTML and XML there will only ever be one `DocumentType`p24 node descendant of the `Document`p16 node. [HTML]p33 [XML]p33

The **`implementation`** attribute must return the `DOMImplementation`p20 object that is associated with the `Document`p16 node.

The **`documentElement`** attribute must return the first child of the `Document`p16 node that is an `Element`p22 node, if there is one, or null otherwise.

The **`defaultView`** attribute must return the context nodep5's browsing context[HTML]'s `WindowProxy`[WINDOW] object.

The **`createElement(`*`tagName`*`)`** method must run the following steps:

1. If *tagName* doesn't match the `Name`[XML] production in XML, raise an `INVALID_CHARACTER_ERR`p10 exception and abort these steps.
2. If the context nodep5 is an HTML documentp16, let *localName* be *tagName*, converted to lowercasep6. Otherwise, let *localName* be *tagName*.
3. Return a new `Element`p22 node with no attributes, `namespaceURI`p15 set to the HTML namespacep8, `prefix`p15 set to null, `localName`p15 set to *localName*, and `ownerDocument`p13 set to the context nodep5.

Note: No check is performed that the local name will match the `NCName`[XMLNS] production in Namespaces in XML.

The **`createElementNS(`*`namespaceURI, qualifiedName`*`)`** method must run the following steps:

1. If *qualifiedName* doesn't match the `Name`[XML] production in XML, raise an `INVALID_CHARACTER_ERR`p10 exception and abort these steps.
2. If *qualifiedName* doesn't match the `QName`[XMLNS] production in Namespaces in XML, raise a `NAMESPACE_ERR`p10 exception and abort these steps.
3. If *qualifiedName* contains a U+003E COLON (":") character, then split the string on the colon and let *prefix* be the part before the colon and *localName* the part after the colon. Otherwise, let *prefix* be null and *localName* be *qualifiedName*.
4. If *prefix* is not null and *namespaceURI* is an empty string, raise a `NAMESPACE_ERR`p10 exception and abort these steps.
5. If *prefix* is "xml" and *namespaceURI* is not the XML namespacep8, raise a `NAMESPACE_ERR`p10 exception and abort these steps.
6. If *qualifiedName* or *prefix* is "xmlns" and *namespaceURI* is not the XMLNS namespacep8, raise a `NAMESPACE_ERR`p10 exception and abort these steps.
7. If *namespaceURI* is the XMLNS namespacep8 and neither *qualifiedName* nor *prefix* is "xmlns", raise a `NAMESPACE_ERR`p10 exception and abort these steps.
8. Return a new `Element`p22 node with no attributes, `namespaceURI`p15 set to *namespaceURI*, `prefix`p15 set to *prefix*, `localName`p15 set to *localName*, and `ownerDocument`p13 set to the context nodep5.

The **`createDocumentFragment()`** method must return a new `DocumentFragment`p16 node with its `ownerDocument`p13 set to the context nodep5.

The **`createTextNode(`*`data`*`)`** method must return a new `Text`p25 node with its `data`p24 attribute set to *data* and `ownerDocument`p13 set to the context nodep5.

Note: No check is performed that the text node contains characters that match the `Char`[XML] production in XML.

The **`createComment(data)`** method must return a new `Comment`p25 node with its `data`p24 attribute set to *data* and `ownerDocument`p13 set to the context nodep5.

Note: No check is performed that the comment contains characters that match the `Char`[XML] production in XML or that it contains two adjacent hyphens or ends with a hyphen.

The **`createProcessingInstruction(target, data)`** method must run the following steps:

1. If the context nodep5 is an HTML documentp16, raise a `NOT_SUPPORTED_ERR`p10 exception and abort these steps.
2. If *target* doesn't match the `Name`[XML] production in XML, raise an `INVALID_CHARACTER_ERR`p10 exception and abort these steps.
3. If *data* contains the string "`?>`", raise an `INVALID_CHARACTER_ERR`p10 exception and abort these steps.
4. Return a new processing instructionp24, with *target* as its targetp24 and *data* as its datap24, and whose `ownerDocument`p13 is set to the context nodep5.

Note: No check is performed that the processing instruction target contains "xml" or the colon, or that the data contains characters that match the `Char`[XML] production in XML.

***collection* = *document* . `getElementsByClassName(classes)`** p19

***collection* = *element* . `getElementsByClassName(classes)`** p23

Returns a `NodeList`p26 of the elements in the object on which the method was invoked (a `Document`p16 or an `Element`p22) that have all the classes given by *classes*.

The *classes* argument is interpreted as a space-separated list of classes.

The **`getElementsByTagName(localName)`** method must run the following steps:

1. If *localName* is just a U+002A ASTERISK ("*") character, return a `NodeList`p26 rooted at the context nodep5, whose filter matches only `Element`p22 nodes.
2. Otherwise, if the context nodep5 is an HTML documentp16, return a `NodeList`p26 rooted at the context nodep5, whose filter matches only the following nodes:
    - `Element`p22 nodes in the HTML namespacep8 that have a `localName`p15 case-sensitivelyp6 equal to *localName*, converted to lowercasep6.
    - `Element`p22 nodes, *not* in the HTML namespacep8, that have a `localName`p15 case-sensitivelyp6 equal to *localName*.
3. Otherwise, return a `NodeList`p26 rooted at the context nodep5, whose filter matches only `Element`p22 nodes that have a `localName`p15 case-sensitivelyp6 equal to *localName*.

A new `NodeList`p26 object must be returned each time.

Note: Thus, in an HTML documentp16, `document.getElementsByTagName("FOO")` will match `FOO` elements that aren't in the HTML namespacep8, and `foo` elements that are in the HTML namespacep8, but not `FOO` elements that are in the HTML namespacep8.

The **`getElementsByTagNameNS(namespaceURI, localName)`** method must run the following steps:

1. If both *namespaceURI* and *localName* are just the character U+002A ASTERISK ("*"), return a `NodeList`p26 rooted at the context nodep5, whose filter matches only `Element`p22 nodes.
2. Otherwise, if *namespaceURI* is just the character U+002A ASTERISK ("*"), return a `NodeList`p26 rooted at the context nodep5, whose filter matches only `Element`p22 nodes with the `localName`p15 equal to *localName*.
3. Otherwise, if *localName* is just the character U+002A ASTERISK ("*"), return a `NodeList`p26 rooted at the context nodep5, whose filter matches only `Element`p22 nodes with the `namespaceURI`p15 equal to *namespaceURI*.

4. Otherwise, return a `NodeList`[p26] rooted at the context node[p5], whose filter matches only `Element`[p22] nodes that have a `namespaceURI`[p15] equal to *namespaceURI* and a `localName`[p15] equal to *localName* (both in a case-sensitive[p6] manner).

A new `NodeList`[p26] object must be returned each time.

The **`getElementsByClassName(`*`classNames`*`)`** method takes a string that contains a set of space-separated tokens[p6] representing classes[p22]. When called, the method must return a live[p26] `NodeList`[p26] object containing all the elements in the context node[p5], in tree order[p5], that have all the classes[p22] specified in the *classNames* argument, having obtained the classes[p22] by splitting the string on spaces[p6]. (Duplicates are ignored.) If there are no tokens specified in the argument, then the method must return an empty `NodeList`[p26]. If the document is in quirks mode[p16], then the comparisons for the classes[p22] must be done in an ASCII case-insensitive[p6] manner, otherwise, the comparisons must be done in a case-sensitive[p6] manner.

A new `NodeList`[p26] object must be returned each time.

> Given the following XHTML fragment:
>
> ```
> <div id="example">
>   <p id="p1" class="aaa bbb"/>
>   <p id="p2" class="aaa ccc"/>
>   <p id="p3" class="bbb ccc"/>
> </div>
> ```
>
> A call to `document.getElementById('example').getElementsByClassName('aaa')`[p23] would return a `NodeList`[p26] with the two paragraphs `p1` and `p2` in it.
>
> A call to `getElementsByClassName('ccc bbb')`[p23] would only return one node, however, namely `p3`. A call to `document.getElementById('example').getElementsByClassName('bbb  ccc ')`[p23] would return the same thing.
>
> A call to `getElementsByClassName('aaa,bbb')`[p23] would return no nodes; none of the elements above are in the "aaa,bbb" class.

The **`getElementById(`*`elementId`*`)`** method must return the first `Element`[p22] node, in tree order[p5], in the context node[p5] whose ID[p22] is *elementId*, or null if there is none.

The **`importNode(`*`importedNode, deep`*`)`** method must run the following steps:

1. If the context node[p5] is an XML document[p16], then if *importedNode* or any of its descendant nodes or any of its `attributes`[p13] or any of the `attributes`[p13] of any descendant `Element`[p22] nodes has a `localName`[p15] which either does not match the `Name`[XML] production in XML or contains a U+003A COLON (":") character, raise an `INVALID_STATE_ERR`[p10] and abort these steps.

2. Return a clone[p8] of *importedNode*, with *new ownerDocument* being the context node[p5], and the *clone children* flag set if *deep* is true.

The **`adoptNode(`*`source`*`)`** method must run the following steps:

1. If *source* is a `Document`[p16] node or a `DocumentType`[p24] node, raise a `NOT_SUPPORTED_ERR`[p10] exception and abort these steps.

2. If the context node[p5] is an XML document[p16], then if *source* or any of its descendant nodes or any of its `attributes`[p13] or any of the `attributes`[p13] of any descendant `Element`[p22] nodes has a `localName`[p15] which either does not match the `Name`[XML] production in XML or contains a U+003A COLON (":") character, raise an `INVALID_STATE_ERR`[p10] and abort these steps.

3. If *source* is an `Element`[p22] node, it is affected by a base URL change[p22].

4. If *source*'s `parentNode`[p13] is not null and its `ownerDocument`[p13] isn't equal to the context node[p5], remove *source* from its parent.

5. Set *source*'s `ownerDocument`[p13] to the context node[p5].

6. If *source* is an `Element`[p22] node, set the `ownerDocument`[p13] of all `Attr`[p21] in its `attributes`[p13] to the context node[p5].

7. For each child node of *source*, call `adoptNode`[p19] on the context node[p5], with the child node as its argument.

8. Return *source*.

document **. compatMode**p20

Returns the string "`CSS1Compat`" if the context nodep5 is in no-quirks modep16 or limited-quirks modep16, and "`BackCompat`", if the `Document`p16 is in quirks modep16.

** **inputEncoding**

** **documentURI** Should document.documentURI really exist? be readonly?

The **`compatMode`** IDL attribute must return the literal string "`CSS1Compat`" unless the context nodep5 is in quirks modep16, in which case it must instead return the literal string "`BackCompat`".

4.3.1 Interface *DOMImplementation*p20

User agents must create a new `DOMImplementation`p20 object whenever a new `Document`p16 node is created and associate it with the that `Document`p16 node.

```
interface DOMImplementation {
  boolean hasFeature(in DOMString feature, [TreatNullAs=EmptyString] in DOMString version);

  DocumentType createDocumentType([TreatNullAs=EmptyString] in DOMString qualifiedName, in
DOMString publicId, in DOMString systemId);
  Document createDocument([TreatNullAs=EmptyString] in DOMString namespaceURI,
[TreatNullAs=EmptyString] in DOMString qualifiedName, in DocumentType doctype);
  Document createHTMLDocument(in DOMString title);
};
```

The **`hasFeature(`*`feature`*`, `*`version`*`)`** method must return true if the user agent supportsp7 the (*feature*, *version*) tuple and false otherwise.

The **`createDocumentType(`*`qualifiedName`*`, `*`publicId`*`, `*`systemId`*`)`** method must run the following steps:

1. If *qualifiedName* doesn't match the `Name`[XML] production in XML, raise an `INVALID_CHARACTER_ERR`p10 exception and abort these steps.
2. If *qualifiedName* doesn't match the `NCName`[XMLNS] production in Namespaces in XML, raise a `NAMESPACE_ERR`p10 exception and abort these steps.
3. Return a new document type declarationp24, with *qualifiedName* as its namep24, *publicId* as its public IDp24, and *systemId* as its system IDp24, and with its `ownerDocument`p13 set to null.

Note: No check is performed that the publicId matches the `PublicChar` production in XML or that the systemId doesn't contain both a quotation mark (") and an apostrophe (').

The **`createDocument(`*`namespaceURI`*`, `*`qualifiedName`*`, `*`doctype`*`)`** method must run the following steps:

1. Let *document* be a new `Document`p16 node.
2. Let *element* be null.
3. If *qualifiedName* is not the empty string, set *element* to the result of invoking the `createElementNS`p17 method with the arguments *namespaceURI* and *qualifiedName* on *document*. If that raised an exception, re-raise the same exception and abort these steps.
4. If *doctype* is not null, run the following substeps:
    1. If the *doctype*'s `ownerDocument`p13 is not null, raise a `WRONG_DOCUMENT_ERR`p10 exception and abort the overall set of steps.
    2. Set the *doctype*'s `ownerDocument`p13 to *document*.

        3. Append *doctype* to *document*.

5. If *element* is not null, append *element* to *document*.
6. Return *document*.

** exceptions for createDocument http://lists.w3.org/Archives/Public/www-dom/2005OctDec/0076.html.

HTML documents[p16] can be created using the `createHTMLDocument()`[p21] method.

***doc* = *document*[WINDOW] . `implementation`[p17] . `createHTMLDocument`[p21](*title*)**

Returns a new `Document`[p16], with a basic DOM already constructed with an appropriate `title` element.

The **`createHTMLDocument(`*`title`*`)`** method, when invoked, must run the following steps:

1. Let *doc* be a newly created `Document`[p16] object.
2. Mark *doc* as being an HTML document[p16].
3. Create a new document type declaration[p24], with "`html`" as its name[p24] and with its `ownerDocument`[p13] set to doc. Append the newly created node to *doc*.
4. Create an `html` element in the HTML namespace[p8], and append it to *doc*.
5. Create a `head` element in the HTML namespace[p8], and append it to the `html` element created in the previous step.
6. Create a `title` element in the HTML namespace[p8], and append it to the `head` element created in the previous step.
7. Create a `Text`[p25] node, and set its `data`[p24] attribute to the string given by the method's argument (which could be the empty string). Append it to the `title` element created in the previous step.
8. Create a `body` element in the HTML namespace[p8], and append it to the `html` element created in the earlier step.
9. Return *doc*.

4.4 Interface *Attr*[p21]

```
interface Attr : Node {
  readonly attribute DOMString name;
  readonly attribute boolean specified;
           attribute DOMString value;
  readonly attribute Element ownerElement;
};
```

`Attr`[p21] nodes represent **attributes**. They have a **name** and an **element** associated with them when they are created. `Attr`[p21] nodes are not considered part of the document tree, so their `parentNode`[p13], `previousSibling`[p13] and `nextSibling`[p13] attributes return null. Also, its child nodes can not be manipulated directly through the `insertBefore`[p13], `replaceChild`[p14] and `appendChild`[p14] methods.

The **`name`** attribute must return the name[p21] associated with the context node[p5].

The **`specified`** attribute must return true.

The **`value`** attribute, on getting, must return the same value as the `textContent`[p15] IDL attribute on the context node[p5], and on setting, must act as if the `textContent`[p15] IDL attribute on the context node[p5] had been set to the new value.

The **`ownerElement`** attribute must return the element[p21] associated with the context node[p5].

This specification further defines two special types of attributes[p21]: ID attributes[p22] and class attributes[p22].

ID attributes must have a `value`p21 that contains at least one character and does not contain any space charactersp5. The `value`p21 must be unique amongst all the IDsp22 in the element's home subtreep5.

> ***Note:** For example, the `id`[HTML] attribute in HTML is an ID attributep22, as well as the `id` attributes in MathML and SVG, and the `id` attribute in the XML namespacep8. [HTML]p33 [MATHML]p33 [SVG]p33 [XMLID]p33*

Class attributes must have a `value`p21 that is a set of space-separated tokensp6 representing the various **classes** that the element belongs to.

The classesp22 that an `Element`p22 node has associated with it is the set of all the classesp22 returned when the value of the class attributep22 is split on spacesp6. (Duplicates are ignored.)

> ***Note:** The `class` attributes in HTML, MathML and SVG are all class attributesp22. [HTML]p33 [MATHML]p33 [SVG]p33*

> ***Note:** This specification does not define the namep21 of IDp22 or class attributesp22.*

4.5 Interface *Element*p22

```
interface Element : Node {
  readonly attribute DOMString tagName;

  DOMString? getAttribute(in DOMString name);
  DOMString? getAttributeNS(in DOMString namespaceURI, in DOMString localName);
  void setAttribute(in DOMString name, in DOMString value);
  void setAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName, in DOMString
value);
  void removeAttribute(in DOMString name);
  void removeAttributeNS(in DOMString namespaceURI, in DOMString localName);
  boolean hasAttribute(in DOMString name);
  boolean hasAttributeNS(in DOMString namespaceURI, in DOMString localName);

  NodeList getElementsByTagName(in DOMString name);
  NodeList getElementsByTagNameNS(in DOMString namespaceURI, in DOMString localName);
  NodeList getElementsByClassName(in DOMString classNames);

           attribute HTMLCollection children;
};
```

`Element`p22 nodes can have a **unique identifier (ID)** associated with them. User agents must associate the `value`p21 of all ID attributesp22 in the `Element`p22 node's `attributes`p13 with the `Element`p22 node, unless it contains less than one character or contains any space charactersp5.

Specifications may define **base URL change steps**.

When an `Element`p22 node is **affected by a base URL change**, the user agent must run the base URL change stepsp22, as defined in other applicable specificationsp6.

The **`tagName`** attribute must, on getting, run the following steps:

1. If the context nodep5's `prefix`p15 is not null, let *tagName* be the concatenation of the context nodep5's `prefix`p15, a U+003E COLON (":") character and its `localName`p15. Otherwise, let *tagName* be just the the context nodep5's `localName`p15.
2. If the context nodep5 is in the HTML namespacep8 and its `ownerDocument`p13 is an HTML documentp16, return *tagName*, converted to uppercasep6. Otherwise, return *tagName*.

The **`getAttribute(`*`name`*`)`** method must run the following steps:

1. If the context nodep5 is in the HTML namespacep8 and its `ownerDocument`p13 is an HTML documentp16, let *name* be *name*, converted to lowercasep6.

2. Return the value of the first attribute in the context node[p5]'s `attributes`[p13] whose name[p21] case-sensitively[p6] equals the first argument, in any namespace, if the attribute is present, or null otherwise.

** **getAttributeNS** (might return null; Gecko and WebKit don't)

The **`setAttribute(`*`name`*`, `*`value`*`)`** method must run the following steps:

1. If *name* is empty or *name* doesn't match the `Name`[XML] production in XML, raise an `INVALID_CHARACTER_ERR`[p10] exception and abort these steps.
2. ** Do something about *name* == "xmlns"? Moz bug 315805 **
3. If the context node[p5] is in the HTML namespace[p8] and its `ownerDocument`[p13] is an HTML document[p16], let *name* be *name*, converted to lowercase[p6].
4. If the node doesn't have an attribute whose name[p21] case-sensitively[p6] equals *name*, create an `Attr`[p21] node, with *name* as its name[p21] and the context node[p5] as its element[p21]. Set its `value`[p21] to *value*. Append this node to the context node's[p5] `attributes`[p13], as its last item.
5. Otherwise, set the `value`[p21] of the first attribute in the context node's[p5] `attributes`[p13] whose name[p21] case-sensitively[p6] equals *name*, in any namespace, to *value*.

** **setAttributeNS**

** **removeAttribute**

** **removeAttributeNS**

** **hasAttribute**

** **hasAttributeNS**

The **`getElementsByTagName(`*`name`*`)`** method on the `Element`[p22] interface must return a live[p26] `NodeList`[p26] with the nodes that the `getElementsByTagName`[p18] method would return when called on the context node's[p5] `ownerDocument`[p13] and passed the same argument, excluding any elements that are not descendants of the context node[p5] on which the method was invoked.

A new `NodeList`[p26] object must be returned each time.

The **`getElementsByTagNameNS(`*`namespaceURI`*`, `*`localName`*`)`** method on the `Element`[p22] interface must return a live[p26] `NodeList`[p26] with the nodes that the `getElementsByTagNameNS`[p18] method would return when called on the context node's[p5] `ownerDocument`[p13] and passed the same arguments, excluding any elements that are not descendants of the context node[p5] on which the method was invoked.

A new `NodeList`[p26] object must be returned each time.

The **`getElementsByClassName(`*`classNames`*`)`** method on the `Element`[p22] interface must return a live[p26] `NodeList`[p26] with the nodes that the `getElementsByClassName`[p19] method would return when called on the context node's[p5] `ownerDocument`[p13] and passed the same argument, excluding any elements that are not descendants of the context node[p5] on which the method was invoked.

A new `NodeList`[p26] object must be returned each time.

The **`children`** attribute must return an `HTMLCollection`[p26] collection[p26], rooted at the context node[p5], whose filter matches only `Element`[p22] nodes whose `parentNode`[p13] is the context node[p5]. ** Or a `NodeList`[p26]?

4.6 Interface *DocumentType*[p24]

```
interface DocumentType : Node {
  readonly attribute DOMString name;
  readonly attribute DOMString publicId;
  readonly attribute DOMString systemId;
};
```

`DocumentType`[p24] nodes represent **document type declarations**. They have a **name** and potentially a **public ID**, and a **system ID** associated with them when they are created.

The **`name`** attribute must, on getting, return the context node's[p5] name[p24].

The **`publicId`** attribute must, on getting, return the context node's[p5] public ID[p24], if it has one, or the empty string otherwise.

The **`systemId`** attribute must, on getting, return the context node's[p5] system ID[p24], if it has one, or the empty string otherwise.

4.7 Interface *ProcessingInstruction*[p24]

```
interface ProcessingInstruction : Node {
  readonly attribute DOMString target;
           attribute DOMString data;
};
```

`ProcessingInstruction`[p24] nodes represent **processing instructions**. They have a **target** and **data** associated with them when they are created.

The **target** attribute must, on getting, return the context node's[p5] target[p24].

The **data** attribute must, on getting, return the context node's[p5] data[p24], and on setting, set the context node's[p5] data[p24] to the new value.

4.8 Interface *CharacterData*[p24]

```
interface CharacterData : Node {
  [TreatNullAs=EmptyString] attribute DOMString data;
  readonly attribute unsigned long length;
  DOMString substringData(in unsigned long offset, in unsigned long count);
  void appendData(in DOMString arg);
  void insertData(in unsigned long offset, in DOMString arg);
  void deleteData(in unsigned long offset, in unsigned long count);
  void replaceData(in unsigned long offset, in unsigned long count, in DOMString arg);
};
```

The **`data`** attribute must, on getting, return the data of the node, and on setting, must change the node's data to the new value.

The **`length`** attribute must, on getting, return the number of UTF-16 code units represented by the node's data.

The **`substringData(`*`offset, count`*`)`** method must run the following steps:

1. If *offset* is negative or is greater than the context node[p5]'s `length`[p24], or if *count* is negative, raise an `INDEX_SIZE_ERR`[p10] exception and abort these steps.
2. If *offset*+*count* is greater than the context node[p5]'s `length`[p24], return a `DOMString`[WEBIDL] whose value is the UTF-16 code units from the *offset*th UTF-16 code unit to the end of *data*.
3. Return a `DOMString`[WEBIDL] whose value is the UTF-16 code units from the *offset*th UTF-16 code unit to the *offset*+*count*th UTF-16 code unit in *data*.

** **appendData**

** **insertData**

** **deleteData**

** **replaceData**

4.9 Interface *Text*[p25]

```
interface Text : CharacterData {
  Text splitText(in unsigned long offset);
  readonly attribute DOMString wholeText;
  Text replaceWholeText(in DOMString content);
};
```

** **splitText**

** **wholeText**

** **replaceWholeText**

4.10 Interface *Comment*[p25]

```
interface Comment : CharacterData {
};
```

5 Collections

A **collection** is an object that represents a lists of DOM nodes. A collection[p26] can be either **live** or **static**. Unless otherwise stated, a collection[p26] must be live[p26].

If a collection[p26] is live[p26], then the attributes and methods on that object must operate on the actual underlying data, not a snapshot of the data.

When a collection[p26] is created, a filter and a root are associated with it.

The collection[p26] then **represents** a view of the subtree rooted at the collection's[p26] root, containing only nodes that match the given filter. The view is linear. In the absence of specific requirements to the contrary, the nodes within the collection[p26] must be sorted in tree order[p5].

An attribute that returns a live[p26] collection[p26] must return the same object every time it is retrieved.

5.1 Interface *NodeList*[p26]

A `NodeList`[p26] object is a kind of collection[p26].

```
interface NodeList {
  getter Node item(in unsigned long index);
  readonly attribute unsigned long length;
};
```

The **`item(index)`** method must return the *index*th node in the collection[p26]. If there is no *index*th node in the collection[p26], then the method must return null.

The **`length`** attribute must, on getting, return the number of nodes represented by the collection[p26].

** `NodeList`[p26]s are enumerable. Explain? `for ... in`

5.2 Interface *HTMLCollection*[p26]

The `HTMLCollection`[p26] interface represents a generic collection[p26] of elements.

Note: This interface is called `HTMLCollection`[p26] for historical reasons. The various getters on this interface return `object`[WEBIDL] for interfaces that inherit from it, which return other objects for historical reasons.

```
interface HTMLCollection {
readonly attribute unsigned long length;
caller getter object item(in unsigned long index); // only returns Element
caller getter object namedItem(in DOMString name); // only returns Element
};
```

***collection* . `length`[p27]**

Returns the number of elements in the collection.

***element* = *collection* . `item`[p27](*index*)**

***collection*[*index*]**

***collection*(*index*)**

Returns the item with index *index* from the collection. The items are sorted in tree order[p5].

Returns null if *index* is out of range.

***element* = *collection* . `namedItem`[p27](*name*)**
***collection*[*name*]**
***collection*(*name*)**

Returns the first item with ID[p22] or name *name* from the collection.

Returns null if no element with that ID[p22] or name could be found.

Only `a`, `applet`, `area`, `embed`, `form`, `frame`, `frameset`, `iframe`, `img`, and `object` elements in the HTML namespace[p8] can have a name for the purpose of this method; their name is given by the value of their `name` attribute.

The object's supported property indices[WEBIDL] are the numbers in the range zero to one less than the number of nodes represented by the collection[p26]. If there are no such elements, then there are no supported property indices[WEBIDL].

The **`length`** attribute must return the number of nodes represented by the collection[p26].

The **`item(index)`** method must return the *index*th node in the collection. If there is no *index*th node in the collection, then the method must return null.

The supported property names[WEBIDL] consist of the values of the `name` attributes of each `a`, `applet`, `area`, `embed`, `form`, `frame`, `frameset`, `iframe`, `img`, and `object` element in the HTML namespace[p8], represented by the collection[p26] with a `name` attribute, plus the list of IDs[p22] that the elements represented by the collection[p26] have.

The **`namedItem(key)`** method must return the first node in the collection[p26] that matches the following requirements:

- It is an `a`, `applet`, `area`, `embed`, `form`, `frame`, `frameset`, `iframe`, `img`, or `object` element, in the HTML namespace[p8], with a `name` attribute equal to *key*, or,

- It is an element with an ID[p22] equal to *key*.

If no such elements are found, then the method must return null.

5.3 Interface *NamedNodeMap*[p27]

A `NamedNodeMap`[p27] object is a kind of collection[p26], whose primary purpose is to expose `Node`[p12]s by name.

```
interface NamedNodeMap {
  Node getNamedItem(in DOMString name);
  Node setNamedItem(in Node arg);
  Node removeNamedItem(in DOMString name);
  Node item(in unsigned long index);
  readonly attribute unsigned long length;
  Node getNamedItemNS(in DOMString namespaceURI, in DOMString localName);
  Node setNamedItemNS(in Node arg);
  Node removeNamedItemNS(in DOMString namespaceURI, in DOMString localName);
};
```

** **getNamedItem**

** **setNamedItem**

** **removeNamedItem**

** **item**

The **`length`** attribute must, on getting, return the number of nodes represented by the collection[p26].

** **getNamedItemNS**

** **setNamedItemNS**

** **removeNamedItemNS**

6 Lists

6.1 Interface *DOMStringList*[p29]

```
interface DOMStringList {
  DOMString item(in unsigned long index);
  readonly attribute unsigned long length;
  boolean contains(in DOMString str);
};
```

** **item**

** **length**

** **contains**

6.2 Interface *DOMTokenList*[p29]

The `DOMTokenList`[p29] interface represents an interface to an underlying string that consists of a set of space-separated tokens[p6].

Note: `DOMTokenList`[p29] objects are always case-sensitive[p6], even when the underlying string might ordinarily be treated in a case-insensitive manner.

```
interface DOMTokenList {
  readonly attribute unsigned long length;
  getter DOMString item(in unsigned long index);
  boolean contains(in DOMString token);
  void add(in DOMString token);
  void remove(in DOMString token);
  boolean toggle(in DOMString token);
  stringifier DOMString ();
};
```

***tokenlist* . `length`[p30]**

Returns the number of tokens in the string.

***element* = *tokenlist* . `item`[p30](*index*)**
***tokenlist*[*index*]**

Returns the token with index *index*. The tokens are returned in the order they are found in the underlying string.

Returns null if *index* is out of range.

***hastoken* = *tokenlist* . `contains`[p30](*token*)**

Returns true if the *token* is present; false otherwise.

Throws a `SYNTAX_ERR`[p10] exception if *token* is empty.

Throws an `INVALID_CHARACTER_ERR`[p10] exception if *token* contains any spaces.

***tokenlist* . `add`[p30](*token*)**

Adds *token*, unless it is already present.

Throws a `SYNTAX_ERR`[p10] exception if *token* is empty.

Throws an `INVALID_CHARACTER_ERR`[p10] exception if *token* contains any spaces.

***tokenlist* . `remove`p30(*token*)**

Removes *token* if it is present.

Throws a `SYNTAX_ERR`p10 exception if *token* is empty.

Throws an `INVALID_CHARACTER_ERR`p10 exception if *token* contains any spaces.

***hastoken* = *tokenlist* . `toggle`p30(*token*)**

Adds *token* if it is not present, or removes it if it is. Returns true if *token* is now present (it was added); returns false if it is not (it was removed).

Throws a `SYNTAX_ERR`p10 exception if *token* is empty.

Throws an `INVALID_CHARACTER_ERR`p10 exception if *token* contains any spaces.

The **`length`** attribute must return the number of tokens that result from splitting the underlying string on spacesp6. This is the *length*p30.

The object's supported property indices[WEBIDL] are the numbers in the range zero to *length*p30−1, unless the *length*p30 is zero, in which case there are no supported property indices[WEBIDL].

The **`item(`*`index`*`)`** method must split the underlying string on spacesp6, preserving the order of the tokens as found in the underlying string, and then return the *index*th item in this list. If *index* is equal to or greater than the number of tokens, then the method must return null.

> For example, if the string is "`a b a c`" then there are four tokens: the token with index 0 is "`a`", the token with index 1 is "`b`", the token with index 2 is "`a`", and the token with index 3 is "`c`".

The **`contains(`*`token`*`)`** method must run the following algorithm:

1. If the *token* argument is the empty string, then raise a `SYNTAX_ERR`p10 exception and stop the algorithm.
2. If the *token* argument contains any space charactersp5, then raise an `INVALID_CHARACTER_ERR`p10 exception and stop the algorithm.
3. Otherwise, split the underlying string on spacesp6 to get the list of tokens in the object's underlying string.
4. If the token indicated by *token* is a case-sensitivep6 match for one of the tokens in the object's underlying string then return true and stop this algorithm.
5. Otherwise, return false.

The **`add(`*`token`*`)`** method must run the following algorithm:

1. If the *token* argument is the empty string, then raise a `SYNTAX_ERR`p10 exception and stop the algorithm.
2. If the *token* argument contains any space charactersp5, then raise an `INVALID_CHARACTER_ERR`p10 exception and stop the algorithm.
3. Otherwise, split the underlying string on spacesp6 to get the list of tokens in the object's underlying string.
4. If the given *token* is a case-sensitivep6 match for one of the tokens in the `DOMTokenList`p29 object's underlying string then stop the algorithm.
5. Otherwise, if the `DOMTokenList`p29 object's underlying string is not the empty string and the last character of that string is not a space characterp5, then append a U+0020 SPACE character to the end of that string.
6. Append the value of *token* to the end of the `DOMTokenList`p29 object's underlying string.

The **`remove(`*`token`*`)`** method must run the following algorithm:

1. If the *token* argument is the empty string, then raise a `SYNTAX_ERR`p10 exception and stop the algorithm.
2. If the *token* argument contains any space charactersp5, then raise an `INVALID_CHARACTER_ERR`p10 exception and stop the algorithm.
3. Otherwise, remove the given *token* from the underlying stringp7.

The **`toggle(`*`token`*`)`** method must run the following algorithm:

1. If the *token* argument is the empty string, then raise a `SYNTAX_ERR` p10 exception and stop the algorithm.
2. If the *token* argument contains any space characters p5, then raise an `INVALID_CHARACTER_ERR` p10 exception and stop the algorithm.
3. Otherwise, split the underlying string on spaces p6 to get the list of tokens in the object's underlying string.
4. If the given *token* is a case-sensitive p6 match for one of the tokens in the `DOMTokenList` p29 object's underlying string then remove the given *token* from the underlying string p7 and stop the algorithm, returning false.
5. Otherwise, if the `DOMTokenList` p29 object's underlying string is not the empty string and the last character of that string is not a space character p5, then append a U+0020 SPACE character to the end of that string.
6. Append the value of *token* to the end of the `DOMTokenList` p29 object's underlying string.
7. Return true.

Objects implementing the `DOMTokenList` p29 interface must **stringify** to the object's underlying string representation.

6.3 Interface *DOMSettableTokenList* p31

The `DOMSettableTokenList` p31 interface is the same as the `DOMTokenList` p29 interface, except that it allows the underlying string to be directly changed.

```
interface DOMSettableTokenList : DOMTokenList {
            attribute DOMString value;
};
```

***tokenlist* . `value`** p31

Returns the underlying string.

Can be set, to change the underlying string.

An object implementing the `DOMSettableTokenList` p31 interface must act as defined for the `DOMTokenList` p29 interface, except for the `value` p31 attribute defined here.

The **`value`** attribute must return the underlying string on getting, and must replace the underlying string with the new value on setting.

7 Historical interfaces

This specification does not define the following interfaces:

- `DOMUserData`
- `DOMObject`
- `NameList`
- `DOMImplementationList`
- `DOMImplementationSource`
- `TypeInfo`
- `UserDataHandler`
- `DOMError`
- `DOMErrorHandler`
- `DOMLocator`
- `DOMConfiguration`
- `CDATASection`
- `Notation`
- `Entity`
- `EntityReference`

References

All references are normative unless marked "Non-normative".

[HTML]

HTML, I. Hickson. WHATWG.

[MATHML]

(Non-normative) *Mathematical Markup Language (MathML)*, D. Carlisle, P. Ion, R. Miner, N. Poppelier. W3C.

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner. IETF.

[SVG]

(Non-normative) *Scalable Vector Graphics (SVG)*, O. Andersson, R. Berjon, E. Dahlström, A. Emmons, J. Ferraiolo, A. Grasso, V. Hardy, S. Hayman, D. Jackson, C. Lilley, C. McCormack, A. Neumann, C. Northway, A. Quint, N. Ramani, D. Schepers, A. Shellshear. W3C.

[WEBIDL]

Web IDL, C. McCormack, S. Weinig. W3C.

[XML]

Extensible Markup Language, T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau. W3C.

[XMLID]

(Non-normative) *xml:id*, J. Marsh, D. Veillard, N. Walsh. W3C.

[XMLNS]

Namespaces in XML, T. Bray, D. Hollander, A. Layman, R. Tobin. W3C.

Acknowledgements

Thanks to Anne van Kesteren, Dethe Elza, and Henri Sivonen, for their useful comments.

Special thanks to Ian Hickson for first specifying some parts of this specification in HTML. [HTML][p33]