**WD-DOM-19980720**

# Level 1 Document Object Model Specification

## Version 1.0

## W3C Working Draft *20 July, 1998*

### This version

http://www.w3.org/TR/1998/WD-DOM-19980720
http://www.w3.org/TR/1998/WD-DOM-19980720/DOM.ps
http://www.w3.org/TR/1998/WD-DOM-19980720/DOM.tgz
http://www.w3.org/TR/1998/WD-DOM-19980720/DOM.zip

### Previous versions

http://www.w3.org/TR/1998/WD-DOM-19980416
http://www.w3.org/TR/WD-DOM-19980318
http://www.w3.org/TR/WD-DOM-971209
http://www.w3.org/TR/WD-DOM-971009

### Latest version

http://www.w3.org/TR/WD-DOM

### WG Chair

Lauren Wood, *SoftQuad, Inc.*

### Editors

Vidur Apparao, *Netscape*
Steve Byrne, *JavaSoft*
Mike Champion, *ArborText*
Scott Isaacs, *Microsoft*
Gavin Nicol, *Inso EPS*
Jonathan Robie, *Texcel Research*
Robert Sutor, *IBM*
Chris Wilson, *Microsoft*

Lauren Wood, *SoftQuad, Inc.*

### Principal Contributors

Vidur Apparao, *Netscape*
Steve Byrne, *JavaSoft (until November 1997)*
Mike Champion, *ArborText, Inc.*
Scott Isaacs, *Microsoft (until January, 1998)*
Arnaud Le Hors, *W3C*
Gavin Nicol, *Inso EPS*
Jonathan Robie, *Texcel Research*
Peter Sharpe, *SoftQuad, Inc.*
Bill Smith, *Sun (after November 1997)*
Jared Sorensen, *Novell*
Robert Sutor, *IBM*
Ray Whitmer, *iMall*
Chris Wilson, *Microsoft (after January, 1998)*

---

# Status

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or members of the DOM working group.

This document has been produced as part of the W3C DOM Activity , and is intended as a draft of a proposed recommendation for the Document Object Model. The authors of this document are the DOM WG members. Different chapters of the Document Object Model specification may have different editors.

We will update this draft specification on a regular basis.

The DOM WG believes that the specification is very close to being ready for Proposed Recommendation status.

Please send detailed comments on this document to w3c-dom-ig@w3.org . We cannot guarantee a personal response but we will try when it is appropriate. Public discussion about the DOM takes place on the www-dom@w3.org mailing list.

The list of known errors in this document is found at http://www.w3.org/DOM/1998/WD-DOM-19980720-errata.html .

---

# Abstract

This specification defines Level 1 of the Document Object Model, a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

The goal of the DOM specification is to define a programmatic interface for XML and HTML. The Level 1 DOM specification is separated into two parts: Core and HTML. The Level 1 Core DOM specification provides a low-level set of fundamental interfaces that can represent any structured document, as well as defining extended interfaces for representing an XML document. These extended XML interfaces need not be implemented by a DOM implementation that only provides access to HTML documents; all of the fundamental interfaces in the Core document must be implemented. A compliant DOM implementation that implements the extended XML interfaces is required to also implement the fundamental Core interfaces, but not the HTML interfaces. The Level 1 HTML specification provides additional, higher-level interfaces that are used with the fundamental interfaces defined in the Level 1 Core specification to provide a more convenient view of an HTML document. A compliant implementation of the HTML DOM will implement all of the fundamental Core interfaces as well as the HTML interfaces.

# Languages used

English, OMG IDL, Java, ECMAScript

# Table of contents

- Expanded Table of Contents [p.5]
- What is the Document Object Model? [p.7]

- Chapter 1: Document Object Model (Core) Level 1 [p.13]
- Chapter 2: Document Object Model (HTML) Level 1 [p.41]

- Appendix A: Production Notes [p.95]
- Appendix B: Document Object Model Contributors [p.99]
- Appendix C: Document Object Model Glossary [p.101]
- Appendix D: Document Object Model Level 1 Bindings [p.107]
- Appendix E: Index of Objects Defined by the DOM [p.177]
- Appendix F: Index of Document Object Model Level 1 Definitions [p.179]

Table of contents

# Expanded Table of Contents


- Expanded Table of Contents [p.5]
- What is the Document Object Model? [p.7]
  - Introduction [p.8]
  - What the Document Object Model is [p.11]
  - What the Document Object Model is not [p.11]
  - Where the Document Object Model came from [p.11]
  - Entities and the DOM Core [p.11]
  - DOM Interfaces and DOM Implementations [p.11]
  - Limitations of Level One [p.11]

- Chapter 1: Document Object Model (Core) Level 1 [p.13]
  - 1.1. Introduction [p.14]
  - 1.2. Fundamental Interfaces [p.14]
  - 1.3. Extended Interfaces [p.37]
- Chapter 2: Document Object Model (HTML) Level 1 [p.41]
  - 2.1. Introduction [p.42]
  - 2.2. HTML Application of Core DOM [p.42]
    - 2.2.1. Naming Conventions [p.43]
  - 2.3. Miscellaneous Object Definitions [p.43]
  - 2.4. Objects related to HTML documents [p.44]
  - 2.5. HTML Elements [p.48]
    - 2.5.1. Property Attributes [p.48]
    - 2.5.2. Naming Exceptions [p.48]
    - 2.5.3. Exposing Element Type Names (tagName) [p.48]
    - 2.5.4. The HTMLElement interface [p.48]
    - 2.5.5. Object definitions [p.49]

- Appendix A: Production Notes [p.95]
  - A.1. The Document Type Definition [p.96]
  - A.2. The production process [p.96]
  - A.3. Object Definitions [p.96]
- Appendix B: Document Object Model Contributors [p.99]
- Appendix C: Document Object Model Glossary [p.101]
- Appendix D: Document Object Model Level 1 Bindings [p.107]
  - D.1. Document Object Model Level 1 Core Bindings [p.107]
    - D.1.1. OMG IDL Definitions for Level 1 Core [p.107]
    - D.1.2. Java Definitions for Level 1 Core [p.110]
    - D.1.3. ECMA Script Definitions for Level 1 Core [p.114]
  - D.2. Document Object Model Level 1 HTML Bindings [p.120]
    - D.2.1. OMG IDL Definitions for the Level 1 HTML DOM [p.120]
    - D.2.2. Java Definitions for the Level 1 HTML DOM [p.130]

- D.2.3. ECMA Script Definitions for the Level 1 HTML DOM [p.149]
- Appendix E: Index of Objects Defined by the DOM [p.177]
- Appendix F: Index of Document Object Model Level 1 Definitions [p.179]

# What is the Document Object Model?

*Editors*
Jonathan Robie, Texcel Research

# Introduction

The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

With the Document Object Model, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM interfaces for the internal subset and external subset have not yet been specified.

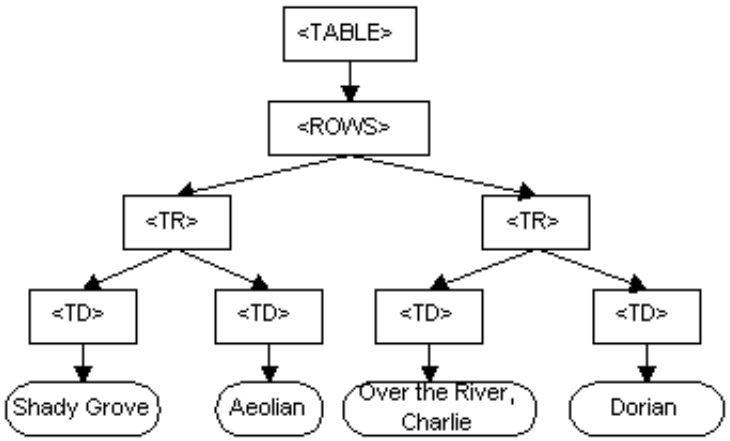
As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The Document Object Model can be used with any programming language. In order to provide precise, language-independent specification of the Document Object Model interfaces, we have chosen to define the specifications in OMG IDL, as defined in the CORBA 2.2 specification. In addition to the OMG IDL specification, we provide language bindings for Java and ECMAScript (an industry-standard scripting language based on JavaScript and JScript). ***Note:*** *OMG IDL is used only as a language-independent and implementation-neutral way to specify interfaces. Various other IDLs could have been used; the use of OMG IDL does not imply a requirement to use a specific object binding runtime.*

# What the Document Object Model is

The Document Object Model is a programming API for documents. The object model itself closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:

```
<TABLE>
<ROWS>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</ROWS>
</TABLE>
```

The Document Object Model represents this table like this:




**DOM representation of the example table**

In the Document Object Model, documents have a logical structure which is very much like a tree; to be more precise, it is like a "forest" or "grove" which can contain more than one tree. However, the Document Object Model does not specify that documents be *implemented* as a tree or a grove , nor does it specify how the relationships among objects be implemented in any way. In other words, the object model specifies the logical model for the programming interface, and this logical model may be implemented in any way that a particular implementation finds convenient. In this specification, we use the term *structure model* to describe the tree-like representation of a document; we specifically avoid terms like "tree" or "grove" in order to avoid implying a particular implementation. One important property of DOM structure models is *structural isomorphism*: if any two Document Object Model implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.

The name "Document Object Model" was chosen because it is an "object model" is used in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity. As an object model, the Document Object Model identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects - including both behavior and attributes
- the relationships and collaborations among these interfaces and objects

The structure of SGML documents has traditionally been represented by an abstract data model, not by an object model. In an abstract data model, the model is centered around the data. In object oriented programming languages, the data itself is encapsulated in objects which hide the data, protecting it from direct external manipulation. The functions associated with these objects determine how the objects may be manipulated, and they are part of the object model.

The Document Object Model currently consists of two parts, DOM Core and DOM HTML. The DOM Core represents the functionality used for XML documents, and also serves as the basis for DOM HTML. All DOM implementations must support the interfaces listed as "fundamental" in the Core specification; in addition, XML implementations must support the interfaces listed as "extended" in the Core specification. The Level 1 DOM HTML specification defines additional functionality needed for HTML documents.

# What the Document Object Model is not

This section is designed to give a more precise understanding of the Document Object Model by distinguishing it from other systems that may seem to be like it.

- Although the Document Object Model was strongly influenced by Dynamic HTML, in Level 1, it does not implement all of Dynamic HTML. In particular, events have not yet been defined. Level 1 is designed to lay a firm foundation for this kind of functionality by providing a robust, flexible model of the document itself.
- The Document Object Model is not a binary specification. Document Object Model programs written in the same language will be source code compatible across platforms, but the Document Object Model does not define any form of binary interoperability.
- The Document Object Model is not a way of persisting objects to XML or HTML. Instead of specifying how objects may be represented in XML, the Document Object Model specifies how XML and HTML documents are represented as objects, so that they may be used in object oriented programs.
- The Document Object Model is not a set of data structures, it is an object model that specifies interfaces. Although this document contains diagrams showing parent/child relationships, these are logical relationships defined by the programming interfaces, not representations of any particular internal data structures.
- The Document Object Model does not define "the true inner semantics" of XML or HTML. The semantics of those languages are defined by the languages themselves. The Document Object Model is a programming model designed to respect these semantics. The Document Object Model does not have any ramifications for the way you write XML and HTML documents; any document that can be written in these languages can be represented in the Document Object Model.
- The Document Object Model, despite its name, is not a competitor to the Component Object Model (COM). COM, like CORBA, is a language independent way to specify interfaces and objects; the Document Object Model is a set of interfaces and objects designed for managing HTML and XML documents. The DOM may be implemented using language-independent systems like COM or CORBA; it may also be implemented using language-specific bindings like the Java or ECMAScript bindings specified in this document.

# Where the Document Object Model came from

The Document Object Model originated as a specification to allow JavaScript scripts and Java programs to be portable among web browsers. Dynamic HTML was the immediate ancestor of the Document Object Model, and it was originally thought of largely in terms of browsers. However, when the Document Object Model Working Group was formed, it was also joined by vendors in other domains, including

HTML or XML editors and document repositories. Several of these vendors had worked with SGML before XML was developed; as a result, the Document Object Model has been influenced by SGML Groves and the HyTime standard. Some of these vendors had also developed their own object models for documents in order to provide programming APIs for SGML/XML editors or document repositories, and these object models have also influenced the Document Object Model.

# Entities and the DOM Core

In the fundamental DOM interfaces, there are no objects representing entities. Numeric character references, and references to the pre-defined entities in HTML and XML, are replaced by the single character that makes up the entity's replacement. For example, in:

```
<p>This is a dog &amp;amp; a cat</p>
```

the "&" will be replaced by the character "&", and the text in the <p> element will form a single continuous sequence of characters. The representation of general entities, both internal and external, are defined within the extended (XML) interfaces of the Level 1 specification. Note: When a DOM representation of a document is serialized as XML or HTML text, applications will need to check each character in text data to see if it needs to be escaped using a numeric or pre-defined entity. Failing to do so could result in invalid HTML or XML.

# DOM Interfaces and DOM Implementations

The DOM specifies interfaces which may be used to manage XML or HTML documents. It is important to realize that these interfaces are an abstraction - much like "abstract base classes" in C++, they are a means of specifying a way to access and manipulate an application's internal representation of a document. In particular, interfaces do not imply a particular concrete implementation. Each DOM application is free to maintain documents in any convenient representation, as long as the interfaces shown in this specification are supported. Some DOM implementations will be existing programs that use the DOM interfaces to access software written long before the DOM specification existed. Therefore, the DOM is designed to avoid implementation dependencies; in particular,

1. Attributes defined in the IDL do not imply concrete objects which must have specific data members - in the language bindings, they are translated to a pair of get()/set() functions, not to a data member. (Read-only functions have only a get() function in the language bindings).
2. DOM applications may provide additional interfaces and objects not found in this specification and still be considered DOM compliant.
3. Because we specify interfaces and not the actual objects that are to be created, the DOM can not know what constructors to call for an implementation. In general, DOM users call the createXXX() methods on the Document class to create document structures, and DOM implementations create their own internal representations of these structures in their implementations of the createXXX() functions.

# Limitations of Level One

The DOM Level 1 specification is intentionally limited to those methods needed to represent and manipulate document structure and content. Future Levels of the DOM specification will provide:

1. A structure model for the internal subset and the external subset.
2. Validation against a schema.
3. Control for rendering documents via stylesheets.
4. Access control.
5. Thread-safety.

# 1. Document Object Model (Core) Level 1

*Editors*

Mike Champion, ArborText (from November 20, 1997)
Steve Byrne, JavaSoft (until November 19, 1997)
Gavin Nicol, Inso EPS

# 1.1. Introduction

This specification defines a minimal set of objects and interfaces for accessing and manipulating document objects. The functionality specified in this draft (the *core* functionality) should be sufficient to implement higher level operations, such as navigating and modifying a document.

Level 1 of the Document Object Model (DOM) provides a common API for software developers and web script authors to access and manipulate parsed HTML and XML content inside conforming products. Primary document structures and some document type declarations are made available. Level 1 also allows creation from scratch of entire web documents in memory; saving those documents persistently is left to the programmer. DOM Level 1 is intentionally limited in scope to methods to represent and manipulate document structure and content; a standard API for controlling how documents are rendered via stylesheets, validated against a DTD, accessed by given users, etc., will be specified in a subsequent revision level of the DOM.

# 1.2. Fundamental Interfaces

The interfaces within this section are considered *fundamental*, and must be fully implemented by all conformant implementations of the DOM, including all HTML DOM implementations.

## Enumeration ***ExceptionCode***

An integer indicating the type of error generated.

### Enumerator Values

| | |
|---|---|
| UNSUPPORTED_DOCUMENT_ERR | If the implementation does not support the type of document requested |
| NOT_CHILD_ERR | If specified node is not a child of this node |
| NO_CHILDREN_ALLOWED_ERR | If this node is of a type that does not allow children of the type of node specified. |
| INDEX_SIZE_ERR | If index is negative, or greater than or equal to the number of nodes in the list |
| WSTRING_SIZE_ERR | If the specfied range of text will not fit into a wstring |
| DATA_SIZE_ERR | If the specfied offset is greater than the number of characters in data |

### Exception *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). In general, DOM methods return specific error values in ordinary processing situation, such as out-of-bound errors when using `NodeList`.

**IDL Definition**

```
exception DOMException {
   ExceptionCode   code;
};
```

### 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. ***Note:*** *The DOM Level 1 does not specify a way of creating a document instance, and hence document creation is an operation specific to an implementation. Future Levels of the DOM specification will provide methods for creating documents directly.*

**IDL Definition**

```
interface DOMImplementation {
  boolean                  hasFeature(in wstring feature,
                                      in wstring version);
};
```

**Methods**

#### *hasFeature*

Test if the DOM implementation implements a specific feature.

**Parameters**

| | |
|---|---|
| **feature** | The package name of the feature to test. In Level 1, the legal values are "HTML" and "XML" (case-insensitive). |
| **version** | This is the version number of the package name to test. If not specified, any supported version of the feature will cause the method to return `TRUE`. |

**Return Values**

`TRUE` if the feature is implemented in the specified version, `FALSE` otherwise.

This method raises no exceptions.

### Interface *DocumentFragment*

`DocumentFragment` is the "lightweight" or "minimal" document object, and it (as the superclass of `Document`) anchors the XML/HTML tree in a full-fledged document.

The children of a `DocumentFragment` node are zero or more nodes representing the tops of any sub-trees defining the structure of the document, including the actual root element of the XML or HTML document, as well as the XML prolog, comments, processing instructions, etc. For a document fragment, there could be a number of subtrees, since fragments 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). Criteria for "well-formedness" are much looser in the HTML world, and the DOM will not attempt to impose any constraints here. For a complete HTML document, this will contain an `Element` instance whose `tagName` is `"HTML"`; for a complete XML document this will contain the outermost element, i.e. the *element* non-terminal in production *[41]* in Section 3 of the XML-lang specification.

DocumentFragments have two primary uses: First of all, a Document IS A DocumentFragment so that a Document really is a tree, and not a forest or grove, with all Nodes having a common ancestor, namely the Document Node itself. For example, a document which started with a comment followed by an HTML element would result in the Document node having two children each with the Document node as parent.

Second of all, 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 itself could fulfill this role, a Document object can potentially be quite heavyweight when what is really needed is a very lightweight object. DocumentFragment is such an object and it contains a reference to a Document so that the Document interface is indirectly available for those times when it is needed.

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

Note that a DocumentFragment must always contain a valid reference to a Document object. If the DocumentFragment *is* a Document, the reference is to itself. A DocumentFragment can contain the same Node types as a Document. However, there is no requirement that a DocumentFragment be a well-formed document. For example, a DocumentFragment might have only one child and that child node could be a TextNode. Such a structure model represents neither an HTML document nor a well-formed XML document. The nature of the DOM's structure model representation means that a DocumentFragment will follow the rules imposed upon a well-formed XML parsed entity. (***ED:*** *The exact relationship between Entity, DocumentFragment and Document is still subject to change.)*

**IDL Definition**

```
interface DocumentFragment : Node {
  readonly attribute  Document            masterDoc;
};
```

**Attributes**

***masterDoc***

This provides access to the `Document` object associated with this `DocumentFragment`.

## Interface *Document*

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. A document *ISA* document fragment that happens to be a discrete unit in the repository, and has contextual information similar to a document transported over HTTP. Since `Document` inherits from `DocumentFragment`, its children are contents of the `Document`, e.g., the root `Element`, the XML prolog, any processing instructions and or comments, etc.

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.

### IDL Definition

```
interface Document : DocumentFragment {
  readonly attribute  DocumentType         doctype;
  readonly attribute  DOMImplementation    implementation;
  readonly attribute  Element              documentElement;
  Element                  createElement(in wstring tagName);
  DocumentFragment         createDocumentFragment();
  Text                     createTextNode(in wstring data);
  Comment                  createComment(in wstring data);
  CDATASection             createCDATASection(in wstring data);
  ProcessingInstruction    createProcessingInstruction(in wstring target,
                                                       in wstring data);
  Attribute                createAttribute(in wstring name);
  Entity                   createEntity();
  EntityReference          createEntityReference();
  NodeList                 getElementsByTagName(in wstring tagname);
};
```

### Attributes

***doctype***

For XML, this provides access to the Document Type Definition (see DocumentType) associated with this XML document. For HTML documents and XML documents without a document type definition this returns `null`.

***implementation***

This provides access to the DOMImplementation object that handles this document. This is necessary because a DOM application may use objects from multiple implementations, and it may be necessary to get a specific document's `DOMImplementation` in order to query the features it supports.

***documentElement***

This is a convenience attribute to that allows direct access to the child node that is the root element of the document. For HTML documents, this is the element with the tagName "HTML".

**Methods**

***createElement***

Create an element of the type specified. Note that the instance returned will implement the Element interface, so attributes can be specified directly on the returned object.

**Parameters**

**tagName** The name of the element type to instantiate.

**Return Values**

A new `Element` object.

This method raises no exceptions.

***createDocumentFragment***

Create an empty `DocumentFragment` object. The masterDoc for this newly created DocumentFragment is the Document on which this method is invoked.

**Return Values**

A new `DocumentFragment`.

This method has no parameters.

This method raises no exceptions.

***createTextNode***

Create a `Text` node given the specified string.

**Parameters**

**data** The data for the node.

**Return Values**

The new `Text` object.

This method raises no exceptions.

***createComment***

Create a `Comment` node given the specified string.

**Parameters**

**data** The data for the node.

**Return Values**

The new `Comment` object.

This method raises no exceptions.

***createCDATASection***

Create a `CDATASection` node whose value is the specified string.
**Parameters**

**data** The data for the CDATASection contents.

**Return Values**

The new `CDATASection` object.
This method raises no exceptions.

***createProcessingInstruction***

Create a `ProcessingInstruction` node given the specified name and data strings.
**Parameters**

**target** The target part of the processing instruction.

**data** The data for the node.

**Return Values**

The new `ProcessingInstruction` object.
This method raises no exceptions.

***createAttribute***

Create an `Attribute` of the given name. Note that the `Attribute` instance can then be set on an `Element` using the `setAttribute` method.
**Parameters**

**name** The name of the attribute.

**Return Values**

A new `Attribute` object.
This method raises no exceptions.

***createEntity***

Creates an Entity object.
**Return Values**

The new `Entity` object.
This method has no parameters.
This method raises no exceptions.

***createEntityReference***

Creates an EntityReference object.
**Return Values**

The new `EntityReference` object.
This method has no parameters.
This method raises no exceptions.

***`getElementsByTagName`***

Returns a collection of all descendant `Elements` with a given tag name.
**Parameters**

| | |
|---|---|
| **tagname** | The name of the tag to match on. If the string "*" is given, this method will return all elements in the document |

**Return Values**

A new `NodeList` object containing reference to all the `Elements` found.
This method raises no exceptions.

## Interface *Node*

The Node object 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 will result in an error.

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), the value returned is null. Note that the specialized interfaces may contain additional and more convenient mechanism to get and set the relevant information.
**IDL Definition**

```
interface Node {
  // NodeType
  const unsigned short      DOCUMENT           = 1;
  const unsigned short      ELEMENT            = 2;
  const unsigned short      ATTRIBUTE          = 3;
  const unsigned short      PROCESSING_INSTRUCTION = 4;
  const unsigned short      COMMENT            = 5;
  const unsigned short      TEXT               = 6;
  const unsigned short      CDATA_SECTION      = 7;
  const unsigned short      DOCUMENT_FRAGMENT  = 8;
  const unsigned short      ENTITY             = 9;
  const unsigned short      ENTITY_REFERENCE   = 10;
  const unsigned short      DOCUMENT_TYPE      = 11;

  readonly attribute  wstring             nodeName;
           attribute  wstring             nodeValue;
```

```
    readonly attribute  unsigned short       nodeType;
    readonly attribute  Node                 parentNode;
    readonly attribute  NodeList             childNodes;
    readonly attribute  Node                 firstChild;
    readonly attribute  Node                 lastChild;
    readonly attribute  Node                 previousSibling;
    readonly attribute  Node                 nextSibling;
    readonly attribute  NamedNodeMap         attributes;
    Node                      insertBefore(in Node newChild,
                                           in Node refChild)
                                           raises(DOMException);
    Node                      replaceChild(in Node newChild,
                                           in Node oldChild)
                                           raises(DOMException);
    Node                      removeChild(in Node oldChild)
                                         raises(DOMException);
    Node                      appendChild(in Node newChild);
    boolean                   hasChildNodes();
    Node                      cloneNode(in boolean deep);
    boolean                   equals(in Node arg,
                                     in boolean deep);
};
```

## Definition group *NodeType*

An integer indicating which type of node this is.
**Defined Constants**

| | |
|---|---|
| **DOCUMENT** | The node is a `Document`. |
| **ELEMENT** | The node is an `Element`. |
| **ATTRIBUTE** | The node is an `Attribute`. |
| **PROCESSING_INSTRUCTION** | The node is a `ProcessingInstruction`. |
| **COMMENT** | The node is a `Comment`. |
| **TEXT** | The node is a `Text`. |
| **CDATA_SECTION** | The node is a `CDATASection`. |
| **DOCUMENT_FRAGMENT** | The node is a `DocumentFragment`. |
| **ENTITY** | The node is an `Entity`. |
| **ENTITY_REFERENCE** | The node is an `EntityReference`. |
| **DOCUMENT_TYPE** | The node is a `DocumentType`. |

(***ED:*** *Should maybe change the order of Nodes to make it more logical.)*

The values of `nodeName`, `nodeValue`, and `attributes` vary according to the node type as follows:

| | **nodeName** | **nodeValue** | **attributes** |
|---|---|---|---|
| Document | #document | null | null |
| Element | tag | null | nodelist |
| Attribute | name | value | null |
| ProcessingInstruction | #processing-instruction | text | null |
| Comment | #comment | comment text | null |
| Text | #text | the text | null |
| CDATASection | #cdata-section | the text | null |
| DocumentFragment | #document-fragment | null | null |
| Entity | #entity | TBD | null |
| EntityReference | #entity-reference | null | null |
| DocumentType | #document-type | TBD | TBD |

**Attributes**

***nodeName***

The name of the node, or a special string for node types that do not have an explicit name associated with them.

***nodeValue***

The value of a node depends on its type; see the table above.

***nodeType***

A code representing the type of the underlying object's type. The actual type of the returned data is dependent on the language binding; the IDL specification uses an `enum`, and it is expected that most language bindings will represent this runtime-queryable node type using an integral data type. The names of the node type enumeration literals are straightforwardly derived from the names of the actual node subtypes, and are fully specified in the IDL definition of node in the OMG IDL Definitions for Level 1 Core [p.107]

***parentNode***

The parent of the given `Node` instance. All nodes, except Documents and DocumentFragments, have a parent. If a node has just been created and not yet added to the tree, it has an implicit parent which is a DocumentFragment. (***ED:*** *Attributes probably do not have a DocumentFragment as a parent. The parent is the Element; maybe it should not be possible to create Attributes outside of the context of the Element that the Attribute belongs to. )*

***childNodes***

A `NodeList` object that will enumerate all children of this node. If there are no children, this is a `NodeList` containing no nodes. The content of the returned `NodeList` is "live" in the sense that changes to the children of the node object that it was created from will be immediately reflected in the nodes returned by the `NodeList` accessors; it is not a static

snapshot of the content of the Node. Similarly, changes made to the nodes returned by the `NodeList` access methods will be immediately reflected in the tree.

***`firstChild`***

The first child of a node. If there is no such node, this is set to `null`.

***`lastChild`***

The last child of a node. If there is no such node, this is set to `null`.

***`previousSibling`***

The node immediately preceding the current node in a breadth-first traversal of the tree. If there is no such node, it is set to `null`.

***`nextSibling`***

The node immediately following the current node in a breadth-first traversal of the tree. If there is no such node, `null` is returned.

***`attributes`***

Provides access to a `NamedNodeMap` containing the node's attributes (if it is an `Element`) or `null` otherwise.

**Methods**

***`insertBefore`***

Inserts a child node `newChild` before the existing child node `refChild`. If `refChild` is `null`, insert `newChild` at the end of the list of children. If `refChild` is not a child of the Node that `insertBefore` is being invoked on, a `DOMException` is raised.

If `newChild` is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node. ***Note:*** *After a successful call to this method, the `newChild` node will be removed from its previous position in the tree, and all `NodeLists` that reference the child list of this object, and previous and next sibling attributes of some children, must be updated.*

**Parameters**

| | |
|---|---|
| **newChild** | The node to insert |
| **refChild** | The reference node, i.e., the node before which the new node will be inserted. |

**Return Values**

The node being inserted.

**Exceptions**

**DOMException**

Thrown if `refChild` is not a child of this node or if this node is of a type that does not allow children of the type of the `newChild` node.

***`replaceChild`***

Replaces the child node `oldChild` with `newChild` in the set of children of the given node, and return the `oldChild` node. If `oldChild` was not already a child of the node that the `replaceChild` method is being invoked on, a `DOMException` is raised. ***Note:*** *After a successful call to this method, the `newChild` node will be removed from its previous position in the tree and all `NodeLists` that reference the child list of this object, and previous and next sibling attributes of some children, must be updated.*

**Parameters**

| | |
|---|---|
| **newChild** | The new node to put in the child list. |
| **oldChild** | The node being replaced in the list. |

**Return Values**

The node replaced.

**Exceptions**

**DOMException**

Thrown if `oldChild` is not a child of the node.

***`removeChild`***

Removes the child node indicated by `oldChild` from the list of children and returns it. If `oldChild` was not a child of the given node, a `DOMException` is raised. ***Note:*** *After a successful call to this method, all `NodeLists` that reference the child list of this object, and previous and next sibling attributes of some children, must be updated.*

**Parameters**

| | |
|---|---|
| **oldChild** | The node being removed |

**Return Values**

The node removed.

**Exceptions**

**DOMException**

Thrown if `oldChild` is not a child of the node.

***`appendChild`***

Adds a child node to the end of the list of children for this node. ***Note:*** *After a successful call to this method, the `newChild` node will be removed from its previous position in the tree and all `NodeLists` that reference the child list of this object, and previous and next sibling attributes of some children, must be updated.*

**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

**Return Values**

The node added.

This method raises no exceptions.

***hasChildNodes***

This is a convenience method to allow easy determination of whether a Node has children or not.

**Return Values**

Set to `true` if the node has any children, `false` if the node has no children at all.

This method has no parameters.

This method raises no exceptions.

***cloneNode***

Returns a duplicate of a given node, i.e., serves as a generic "copy constructor" for Nodes. ***Note:*** *Cloning a Text node will copy the Text Node and the text it contains; cloning an Element will not copy any text it contains unless it is a deep clone, since the text is contained in a Text node.*

**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). ***Note:*** *When Elements are cloned, all attribute nodes are cloned, including those generated by the XML processor to represent defaulted attributes*

**Return Values**

The duplicate node.

This method raises no exceptions.

***equals***

Tests whether two nodes are equivalent. ***Note:*** *This method tests for equality of nodes, not* sameness *(i.e. whether the two nodes are exactly the same object) which should be implemented using language-specific methods. All objects that are the* same *will also be* equal*, though the reverse may not be true.For example, in C++ the == operator could be bound to test for* sameness*. So if two objects were equivalent in terms of == they would also be equivalent in terms of this method, but even if the objects compared true using this method, they may compare differently in terms of the == operator.*

**Parameters**

**arg** The node to compare equality with.

**deep** A flag indicating whether the entire subtree below the object should also be checked for equality. Is this is `true` and the return value is `true`, the two subtree should be considered equivalent.

**Return Values**

If the nodes, and possibly subtrees are equivalent, `true` otherwise `false`.

This method raises no exceptions.

## Interface *NodeList*

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented, allowing different DOM implementations to be tuned for their specific environments.

The items in the `NodeList` are accessible via an integral index, starting from 0.

**IDL Definition**

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

**Methods**

***item***

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of nodes in the list, NULL is returned.

**Parameters**

**index** Index into the collection

**Return Values**

The node at the `index` position in the `NodeList`, or `null` if that is not a valid index.

This method raises no exceptions.

**Attributes**

***size***

The number of nodes in the `NodeList` instance. The range of valid child node indices is 0 to `size-1` inclusive.

## Interface *NamedNodeMap*

Objects implementing the `NamedNodeMap` interface are used to represent collections of nodes that can be accessed by name. Objects contained in an object implementing `NamedNodeMap` may also be accessed by ordinal index. The ability to access members of a NamedNodeMap by ordinal index does not imply that the DOM specifies an order to these Nodes. These methods may simply be used for enumerating all of members of the NamedNodeMap.

**IDL Definition**

```
interface NamedNodeMap {
  Node                    getNamedItem(in wstring name);
  void                    setNamedItem(in Node arg);
  Node                    removeNamedItem(in wstring name);
  Node                    item(in unsigned long index);
  readonly attribute  unsigned long        size;
};
```

**Methods**

***`getNamedItem`***

Retrieves a node from a list by name

**Parameters**

**name** Name of a node to retrieve.

**Return Values**

A `Node` (of any type) with the specified name, or `null` if the specified name did not identify any node in the list.

This method raises no exceptions.

***`setNamedItem`***

Add a node to a `NamedNodeMap` using the `nodeName` attribute of the node.***Note:*** *As the `nodeName` attribute is used to derive the named 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 will clash. This is seen as preferable to allowing nodes to be aliased.*

**Parameters**

**arg** A node to store in a named node list. The node will later be accessible using the value of the `nodeName` attribute of the node.

This method returns nothing.

This method raises no exceptions.

***`removeNamedItem`***

Remove a node identified by its name.

**Parameters**

**name** The name of a node to remove

**Return Values**

The node removed from the list or `null` if no node with such a name exists.

This method raises no exceptions.

***item***

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of nodes in the list, `null` is returned.

**Parameters**

**index** Index into the collection

**Return Values**

The node at the `index` position in the `NamedNodeMap`, or `null` if that is not a valid index.

This method raises no exceptions.

**Attributes**

***size***

The number of nodes in the `NamedNodeMap` instance. The range of valid child node indices is 0 to `size-1` inclusive.

## Interface *Data*

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

**IDL Definition**

```
interface Data : Node {
           attribute  wstring             data;
  readonly attribute  unsigned long       size;
  wstring                   substring(in unsigned long start,
                                      in unsigned long count)
                                      raises(DOMException);
  void                      append(in wstring arg);
  void                      insert(in unsigned long offset,
                                   in wstring arg)
                                   raises(DOMException);
  void                      delete(in unsigned long offset,
                                   in unsigned long count)
                                   raises(DOMException);
  void                      replace(in unsigned long offset,
```

```
                                        in unsigned long count,
                                        in wstring arg)
                                        raises(DOMException);
};
```

***Note:*** *Since Data is a general name, and this interface is designed for Character Data (text), the name of this interface may change. Some method and attribute names may also change to avoid confusion.*

**Attributes**

***`data`***

This provides access to the character data of a node that implements these interfaces. If the character data of node cannot fit into the length of a `wstring` a `DOMException` is raised. If this exception is detected, the user may call `substring` to retrieve the data in manageable chunks.

***`size`***

This provides access to the number of characters that are available through `data` and the `substring` method below. This may have the value zero, i.e., Data nodes may be empty.

**Methods**

***`substring`***

Extracts a range of the data from this object implementing these interfaces.

**Parameters**

| | |
|---|---|
| **start** | Start offset of substring to extract |
| **count** | The number of characters to extract. |

**Return Values**

This method returns the specified substring.

**Exceptions**

**DOMException**

Thrown if the specified range of text will not fit into a `wstring`.

***`append`***

Append the string to the end of the character data in the object implementing these interfaces. Upon success, `data` will provide access to the concatenation of `data` and the `wstring` specified.

**Parameters**

| | |
|---|---|
| **arg** | The `wstring` to append. |

This method returns nothing.
This method raises no exceptions.

***insert***

Insert a string at the specified character offset.
**Parameters**

**offset** The character offset at which to insert

**arg** The `wstring` to insert

**Exceptions**
**DOMException**

Thrown if the specified offset is greater than the number of characters in `data`.
This method returns nothing.

***delete***

Remove a range of characters from the node. Upon success, `data` and `size` will reflect the change.
**Parameters**

**offset** The offset from which to remove characters.

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

**Exceptions**
**DOMException**

Thrown if the specified offset is greater than the number of characters in `data`.
This method returns nothing.

***replace***

Replace the characters starting at the specified character offset with the specified string.
**Parameters**

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

**count** The number of characters to replace. If the sum of `offset` and `count` exceeds `size`, then all characters 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 `wstring` with which the range will be replaced.

**Exceptions**

**DOMException**

Thrown if the specified offset is greater than the number of characters in `data`.

This method returns nothing.

## Interface *Attribute*

The `Attribute` interface represents an attribute in an `Element` object. Typically the allowable values for the attribute are defined in a document type definition.

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 has no effective value. Note, in particular, that an effective value of the `null` string would be returned as a `Text` node instance whose `data` attribute will contain a zero length string. If the attribute has no effective value, then this method will return `null`. Note the `value` attribute on the `Attribute` instance can also be used to retrieve the string version of the attribute's value(s). (***ED:*** *We should probably have a table here showing how the combination of the attribute value and specified suffice to also determine if an attribute was defaulted.)*

The `Attribute` inherits the `Node` interface, which has a `parentNode` attribute. This attribute is set to the `Element` associated with the attribute as an expedient way of getting from the attribute to the `Element`. ***Note:*** *In XML, the value of an attribute is represented by the child nodes of an attribute node, since the value can be contain entity references. Thus, attributes which contain entity references will have a child list containing both text nodes and entity reference nodes. In addition, tokenised attribute types, such as* `NMTOKENS` *will result in a child list where each child represents a single token from the attribute value.*

**IDL Definition**

```
interface Attribute : Node {
  wstring                    getName();
          attribute  boolean              specified;
  wstring                    getValue();
};
```

**Methods**

***getName***

Returns the name of this attribute. (***ED:*** *Given that we have nodeName on node, is this necessary? Also, even if we leave it in, it should probably be a readonly attribute.)*

**Return Values**

(***ED:*** *TBD.)*

This method has no parameters.

This method raises no exceptions.

**Attributes**

***specified***

If this attribute was explicitly given a value in the original document, this will be true; otherwise, it will be false.

**Methods**

***getValue***

Returns the value of the attribute as a string. Character and general entity references will have been replaced with their values in the returned string.

**Return Values**

The value of the attribute as a `wstring`. (***ED:*** *Given that we have nodeValue on node, is this necessary? Also, even if we leave it in, it should probably be an attribute.)*

This method has no parameters.

This method raises no exceptions.

## Interface *Element*

By far the vast majority (apart from text) of node types that authors will encounter when traversing a document will be Element nodes. These objects represent both the element itself, as well as any contained nodes. For example (in XML):

```
<elementExample id="demo">
  <subelement1/>
  <subelement2><subsubelement/></subelement2>
</elementExample>
```

When represented using DOM, the top node would be "elementExample", which contains two child Element nodes (and some space), one for "subelement1" and one for "subelement2". "subelement1" contains no child nodes of its own.

**IDL Definition**

```
interface Element : Node {
  wstring                  getTagName();
  NamedNodeMap             getAttributes();
  wstring                  getAttribute(in wstring name);
  void                     setAttribute(in string name,
                                        in string value);
  void                     removeAttribute(in wstring name);
  Attribute                getAttributeNode(in wstring name);
  void                     setAttributeNode(in Attribute newAttr);
  void                     removeAttributeNode(in Attribute oldAttr);
  NodeList                 getElementsByTagName(in wstring tagname);
  void                     normalize();
};
```

**Methods**

***getTagName***

This method returns the string that is the element's name. For example, in:

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

This would have the value `"elementExample"`. Note that this is case-preserving, as are all of the operations of the DOM. (***ED:*** *Need to add section about name case etc. and a reference to it here. )*
(***ED:*** *Could we choose a better name for nodeName? Should tagName be an attribute (possibly read-only)?)*
**Return Values**

The element's tag name
This method has no parameters.
This method raises no exceptions.

***`getAttributes`***

Returns the attributes for this element. In the `elementExample` example above, the attributes list would consist of the `id` attribute, as well as any attributes which were defined by the document type definition for this element which have default values. (***ED:*** *Is this necessary given that we have attributes() on node? Should this be an attribute here?)*
**Return Values**

Attribute list
This method has no parameters.
This method raises no exceptions.

***`getAttribute`***

Retrieves an `Attribute` value by name.
**Parameters**

| | |
|---|---|
| **name** | The name of the attribute to retrieve |

**Return Values**

The `Attribute` value as a string, or the empty string if that attribute does not have a specified or defaulted value.
This method raises no exceptions.

***`setAttribute`***

Adds a new attribute/value pair. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. (***ED:*** *Perhaps this should return the Attribute so that when a value is being replaced, you can still access it?)*
**Parameters**

**name** Name of an attribute

**value** Value to set in string form

This method returns nothing.
This method raises no exceptions.

***removeAttribute***

Removes the specified attribute. (***ED:*** *Perhaps this should return the Attribute so that when an attribute is being removed, you can still access it?)*
**Parameters**

**name** The name of attribute to remove

This method returns nothing.
This method raises no exceptions.

***getAttributeNode***

Retrieves an attribute node by name.
**Parameters**

**name** The name of the attribute to retrieve

**Return Values**

The attribute node with the specified attribute name
This method raises no exceptions.

***setAttributeNode***

Adds a new attribute/value pair. If an attribute by that name is already present in the element, its value is changed to be that of the Attribute instance.(***ED:*** *Perhaps this should return the Attribute so that when an attribute is being removed, you can still access it?)*
**Parameters**

**newAttr** The attribute node to add to the attribute list

This method returns nothing.
This method raises no exceptions.

***removeAttributeNode***

Removes the specified attribute/value pair.(***ED:*** *Perhaps this should return the Attribute so that when an attribute is being removed, you can still access it?)*
**Parameters**

**oldAttr** The attribute node to remove from attribute list

This method returns nothing.
This method raises no exceptions.

***getElementsByTagName***

Returns a list of all elements in the sub-tree of this element with a given tag name.
**Parameters**

| | |
|---|---|
| **tagname** | The name of the tag to match on, or the wildcard string "*" to return all elements. |

**Return Values**

This method returns a list of element nodes that have the specified tag name.
This method raises no exceptions.

***normalize***

Puts all `Text` nodes in the sub-tree underneath this `Element` into a "normal" form where only markup (e.g., tags, comments, processing instructions, CDATA sections, and entity references separates `Text` nodes. This can be useful to ensure that the DOM view of a document is identical to how it would look if saved and re-loaded, and is useful if operations (such as XPointer lookups) that depend on a particular document tree structure must be allowed.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

## Interface *Text*

The `text` interface represents the non-markup content of an `Element`. If there is no markup inside an element's content, the text will be contained in a single object implementing the `Text` interface that is the child of the element. Any markup will parse into child elements that are siblings of the text nodes on either side of it, and whose content will be represented as text node children of the markup element.

When a document is first made available to 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 `normalize()` method on `Element` will merge any such adjacent Text objects into a single node for each block of text; this is recommended before employing operations that depend on a particular document structure, such as navigation with `XPointers`.
**IDL Definition**

```
interface Text : Data {
  Text                      splitText(in unsigned long offset);
  Text                      joinText(in Text node1,
                                     in Text node2);
};
```

**Methods**

***splitText***

Breaks a text node into two text nodes at the specified offset, keeping both in the tree as siblings.

**Parameters**

**offset** The offset at which to split.

**Return Values**

This method returns the new text node containing all the content at and after the `offset` point. The original node contains all the content up to the `offset` point.

This method raises no exceptions.

***joinText***

Joins the contents of two Text nodes into a single text node, with only one returned node left in the tree.

**Parameters**

**node1** The first Text node to join.

**node2** The second Text node to join.

**Return Values**

This method returns a new text node containing the contents of `node1` and `node2`. The input nodes are not modified.(***ED:*** *This is just a suggestion; the WG has not decided on the actual semantics of what joinText does to the original nodes.)*

This method raises no exceptions.

## Interface *Comment*

This 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.

**IDL Definition**

```
interface Comment : Data {
};
```

### Interface *ProcessingInstruction*

The `ProcessingInstruction` interface represents a "processing instruction", used in XML (and legal, though seldom supported, in HTML) as a way to keep processor-specific information in the text of the document. The content of the node is the entire content between the delimiters of the processing instruction.

**IDL Definition**

```
interface ProcessingInstruction : Node {
           attribute  wstring             target;
           attribute  wstring             data;
};
```

**Attributes**

***target***

XML defines a target as the first token following the markup that begins the processing instruction, and this attribute returns that name. For HTML, the returned value is `null`.

***data***

The content of the processing instruction, from the character immediately after the `<?` (after the target in XML) to the character immediately preceding the `?>` (the > in HTML).(***ED:*** *How does this relate to the attribute on Node?)*

# 1.3. Extended Interfaces

The interfaces defined here form part of the DOM Level 1 Core specification, but objects that expose these interfaces will never be encountered in a DOM implementation that deals only with HTML. As such, HTML-only DOM implementations do not need to have objects that implement these interfaces.

### Interface *CDATASection*

CDATA sections are used in the document instance, and provide a region in which most of the XML delimiter recognition does not take place. The primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The `wstring` attribute of the `Text` node holds the text that was contained by the CDATA section. Note that this *may* contain characters that need to be escaped outside of CDATA sections.

**IDL Definition**

```
interface CDATASection : Text {
};
```

### Interface *DocumentType*

Each document has a (possibly null) attribute that contains a reference to a `DocumentType` object. The `DocumentType` class in the DOM Level 1 core provides an interface to the list of entities that are defined for the document, and little else because the effect of namespaces and the various XML scheme efforts on DTD representation are not yet clearly understood.

### IDL Definition

```
interface DocumentType : Node {
           attribute  wstring             name;
  readonly attribute  NamedNodeMap        entities;
  readonly attribute  NamedNodeMap        notations;
};
```

### Attributes

***name***

The `name` attribute is a `wstring` that holds the name of DTD; i.e. the name immediately following the `DOCTYPE` keyword.

***entities***

This is a `NamedNodeMap` containing the general entities, both external and internal, declared in the DTD. For example in:

```
<!DOCTYPE ex SYSTEM "ex.dtd" [
  <!ENTITY foo "foo">
  <!ENTITY bar "bar">
  <!ENTITY % baz "baz">
]>
<ex/>
```

the interface would provide access to `foo` and `bar` but not `baz`. All objects supporting the `Node` interface that are accessed though this attribute, will also support the `Entity` interface. For HTML, this will always be `null`.

***notations***

This is a `NamedNodeMap` containing the notations declared in the DTD. Each node in this map will also implement the `Notation` interface.

## Interface *Notation*

This interface represents a notation declared in the DTD. A notation declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification). The `nodeName` attribute inherited from `Node` is set to the declared name of the notation.

### IDL Definition

```
interface Notation : Node {
           attribute  wstring             publicId;
           attribute  wstring             systemId;
};
```

### Attributes

***publicId***

The public identifier for the notation. If the public identifier was not specified, this is `null`.

***systemId***

The system identifier for the notation. If the system identifier was not specified, this is `null`.

### Interface *Entity*

This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself *not* the entity declaration. Entity declaration modeling has been left for a later Level of the DOM specification.

The `nodeName` attribute that is inherited from `Node` contains the name of the entity.

The structure of the child list is exactly the same as the structure of the child list for an `EntityReference` with the same `nodeName` value. (***ED:*** *Should we have something about resolution of entities? What happens if an entity was declared but not resolved? Should we have a method to force resolution?)*

**IDL Definition**

```
interface Entity : Node {
           attribute  wstring             publicId;
           attribute  wstring             systemId;
           attribute  wstring             notationName;
};
```

**Attributes**

***publicId***

The public identifier associated with the entity, if specified. If the public identifier was not specified, this is `null`.

***systemId***

The system identifier associated with the entity, if specified. If the system identifier was not specified, this is `null`.

***notationName***

For unparsed entities, the name of the notation for the entity. For parsed entities, this is `null`. (***ED:*** *Perhaps there should be a predefined value for parsed entities other than null?)*

### Interface *EntityReference*

`EntityReference` objects are inserted into the DOM by the XML processor whenever the processor sees a reference to an entity other than the pre-defined character entities in the XML specification. The replacement value, if it is available, will appear in the child list of the `EntityReference`.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available. ***Note:*** *There is a suggestion that the* `nodeName` *attribute inherited from* `Node` *be set to the declared name of the entity. This could then be used to access the entity via the* `NamedNodeMap` *in the object implementing the* `DocumentType` *interfaces for a given document. Another suggestion is to add an* `entityName` *attribute instead, to avoid clashes with the nodeName of the Entity itself.*

**IDL Definition**

```
interface EntityReference : Node {
};
```

# 2. Document Object Model (HTML) Level 1

*Editors*

Mike Champion, ArborText
Gavin Nicol, Inso EPS
Vidur Apparao, Netscape
Scott Isaacs, Microsoft (until January 1998)
Chris Wilson, Microsoft (after January 1998)

# 2.1. Introduction

This specification is part of the W3C Document Object Model [p.1] , which defines a programming interface for manipulating HTML and XML documents. Objects and methods that are used for both HTML and XML documents are defined in the W3C DOM Level 1 Core [p.13] specification, which forms the basis for the current document. This document extends the Level 1 Core specification to describe objects and methods specific to HTML documents. In general, the functionality needed to manipulate hierarchical document structures, elements, and attributes will be found in the core specification; functionality that depends on the specific elements defined in HTML will be found in this document.

The goals of the HTML-specific DOM specification are:

- to specialize and add functionality that relates specifically to HTML documents and elements.
- to address issues of backwards compatibility with the Level 0 Document Object Model.
- to provide convenience mechanisms, where appropriate, for common and frequent operations on HTML documents.

The key differences between the core DOM and the HTML application of DOM is that the HTML Document Object Model exposes a number of convenience methods and properties that are consistent with the existing models and are more appropriate to script writers. In many cases, these enhancements are not applicable to a general DOM because they rely on the presence of a predefined DTD. For DOM Level 1, the transitional and frameset DTDs for HTML 4.0 are assumed.

The W3C DOM Working Group attempts, wherever possible, to maintain compatibility with the DOM Level 0, which is defined as being the functionality exposed in Netscape Navigator 3.0 and Microsoft Internet Explorer 3.0.

More specifically, this document includes the following specializations for HTML:

- An HTMLDocument interface, derived from the core Document interface. HTMLDocument specifies the operations and queries that can be made on a HTML document.
- An HTMLElement interface, derived from the core Element interface. HTMLElement specifies the operations and queries that can be made on any HTML element. Methods on HTMLElement include those that allow for the retrieval and modification of attributes that apply to all HTML elements.
- Specializations for all HTML elements that have attributes that extend beyond those specified in the HTMLElement interface. For all such attributes, the derived interface for the element contains explicit methods for setting and getting the values.

The Level 1 document does not include mechanisms to access and modify style specified through CSS 1. Furthermore, it does not define an event model for HTML documents. This functionality will be specified in a future Level of this specification.

# 2.2. HTML Application of Core DOM

## 2.2.1. Naming Conventions

The HTML DOM follows a naming convention for properties, methods, events, collections, and data types. All names are defined as one or more English words concatenated together to form a single string.
Properties and Methods

The property or method name starts with the initial keyword in lowercase, and each subsequent word starts with a capital letter. For example, a property that returns document meta information such as the date the file was created might be named "fileDateCreated". In the ECMAScript binding, properties are exposed as properties of a given object. In Java, properties are exposed with get and set methods.
Non-HTML 4.0 interfaces and attributes

While most of the interfaces defined below can be mapped directly to elements defined in the HTML 4.0 Recommendation, some of them cannot. Similarly, not all attributes listed below have counterparts in the HTML 4.0 specification (and some do, but have been renamed to avoid conflicts with scripting languages). Interfaces and attribute definitions that have links to the HTML 4.0 specification are defined there; all others are added by this specification, either for convenience or backwards compatibility with DOM Level 0 implementations.

# 2.3. Miscellaneous Object Definitions

**Interface *HTMLCollection***

An `HTMLCollection` is a list of nodes. An individual node may be accessed by either ordinal index or the node's `name` or `id` attributes. *Note:* Collections in the HTML DOM are assumed to *live* meaning that they are automatically updated when the underlying document is changed.

**IDL Definition**

```
interface HTMLCollection {
  readonly attribute  long                length;
  Node                        item(in long index);
  Node                        namedItem(in wstring name);
};
```

**Attributes**

***length***

This attribute specifies the length or *size* of the list.

**Methods**

***item***

This method retrieves a node specified by ordinal index. Nodes are numbered in tree order (depth-first traversal order).

**Parameters**

**index** The index of the node to be fetched.

**Return Values**

The `Node` at the corresponding position upon success. A value of null is returned if the index is out of range.

This method raises no exceptions.

***namedItem***

This method retrieves a `Node` using a name.

**Parameters**

**name** The name of the `Node` to be fetched.

**Return Values**

The `Node` with a `name` or `id` attribute whose value corresponds to the specified string. Upon failure (e.g., no Node with this name exists), returns null.

This method raises no exceptions.

# 2.4. Objects related to HTML documents

### Interface *HTMLDocument*

An `HTMLDocument` is the root of the HTML hierarchy and holds the entire content. Beside providing access to the hierarchy, it also provides some convenience methods for accessing certain sets of information from the document.

The following properties have been deprecated in favor of the corresponding ones for the BODY element:

- alinkColor
- background
- bgColor
- fgColor
- linkColor
- vlinkColor

**IDL Definition**

```
interface HTMLDocument : Document {
           attribute  wstring             title;
  readonly attribute  wstring             referrer;
  readonly attribute  wstring             fileSize;
  readonly attribute  wstring             fileCreatedDate;
  readonly attribute  wstring             fileModifiedDate;
  readonly attribute  wstring             fileUpdatedDate;
  readonly attribute  wstring             domain;
```

```
  readonly attribute  wstring             URL;
           attribute  HTMLElement         body;
  readonly attribute  HTMLCollection      images;
  readonly attribute  HTMLCollection      applets;
  readonly attribute  HTMLCollection      links;
  readonly attribute  HTMLCollection      forms;
  readonly attribute  HTMLCollection      anchors;
           attribute  wstring             cookie;
  void                     open();
  void                     close();
  void                     write(in wstring text);
  void                     writeln(in wstring text);
  Element                  getElementById(in wstring elementId);
  NodeList                 getElementsByName(in wstring elementName);
};
```

### Attributes

***title***

The title of a document as specified by the `TITLE` element in the head of the document.

***referrer***

Returns the URI of the page that linked to this page. The value is an empty string if the user navigated to the page directly (not through a link, but, for example, via a bookmark).

***fileSize***

The size of the document, in bytes.

***fileCreatedDate***

A string representation of the date the document was created. The value is derived from the HTTP header data sent by the Web server. Servers generally obtain this date by examining the file's modification date. The date format is that specified by the HTTP protocol.

***fileModifiedDate***

A string representation of the date the document was last modified. See `fileCreatedDate` for format information.

***fileUpdatedDate***

A string representation of the date the document was last updated. See `fileCreatedDate` for format information.

***domain***

The domain name of the server that served the document, or a null string if the server cannot be identified by a domain name.

***URL***

The complete URI of the document.

***body***

The element that contains the content for the document. In documents with `BODY` contents, returns the `BODY` element, and in frameset documents, this returns the outermost `FRAMESET` element.

***images***

A collection of all the `IMG` elements in a document. The behavior is limited to `IMG` elements for backwards compatibility.

***applets***

A collection of all the `OBJECT` elements that include applets and `APPLET` (*deprecated*) elements in a document.

***links***

A collection of all `AREA` elements and anchor (`A`) elements in a document with a value for the `href` attribute.

***forms***

A collection of all the forms of a document.

***anchors***

A collection of all the anchor (`A`) elements in a document with a value for the `name` attribute.

***cookie***

The cookies associated with this document. If there are none, the value is an empty string. Otherwise, the value is a string: a semicolon-delimited list of "name, value" pairs for all the cookies associated with the page. For example, `name=value;expires=date`.

**Methods**

***open***

*Note.* This method and the ones following allow a user to add to or replace the structure model of a document using strings of unparsed HTML. The Working Group is currently considering alternate methods for providing similar functionality for both HTML and XML documents. The following methods may be deprecated at some point in the future in favor of a more general-purpose mechanism.

Open a document stream for writing. If a document exists in the target, this method clears it.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***close***

Closes a document stream opened by `open()` and forces display.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***write***

Write a string of text to a document stream opened by `open()`. The text is parsed into the document's structure model.

**Parameters**

| | |
|---|---|
| **text** | The string to be parsed into some structure in the document structure model. |

This method returns nothing.
This method raises no exceptions.

***writeln***

Write a string of text followed by a newline character to a document stream opened by `open()`. The text is parsed into the document's structure model.

**Parameters**

| | |
|---|---|
| **text** | The string to be parsed into some structure in the document structure model. |

This method returns nothing.
This method raises no exceptions.

***getElementById***

Returns the Element whose `id` is given by elementId. If no such element exists, returns void. Behavior is not defined if more than one element has this `id`.

**Parameters**

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

**Return Values**

This method raises no exceptions.

***getElementsByName***

Returns the (possibly empty) collection of elements whose `name` value is given by elementName.

**Parameters**

| | |
|---|---|
| **elementName** | The `name` attribute value for an element. |

**Return Values**

This method raises no exceptions.

# 2.5. HTML Elements

## 2.5.1. Property Attributes

HTML attributes are exposed as properties on the element object. The name of the exposed property always uses the naming conventions, and is independent of the case of the attribute in the source document. The data type of the property is determined by the type of the attribute as determined by the HTML 4.0 transitional and frameset DTDs. The attributes have the semantics (including case-sensitivity) given in the HTML 4.0 specification.

The attributes are exposed as properties for compatibility with Level 0. This usage is deprecated because it can not be generalized to all possible attribute names, as is required both for XML and potentially for future versions of HTML. We are considering how best to add, get, and remove attributes in a simple and direct way.

| DTD Data Type | Object Model Data Type |
|---|---|
| CDATA | String |
| Value list (e.g.. (left \| right \| center)) | String |
| one-value Value list (e.g. (border)) | Boolean |
| Number | Long Integer |

The return value of an attribute that has data type that is a value list is always capitalized, independent of the case of the value in the source document. For example, if the value of the align attribute on a P element is "left" then it is returned as "Left". For attributes with the `CDATA` data type, the case of the return value is that given in the source document.

## 2.5.2. Naming Exceptions

To avoid name-space conflicts, an attribute with the same name as a keyword in one of our chosen binding languages will be prefixed. For HTML, the prefix used is "html". For example, the `for` attribute of the `LABEL` element collides with loop construct naming conventions and is renamed `htmlFor`.

## 2.5.3. Exposing Element Type Names (tagName)

The property that exposes the element type name will return this in uppercase. For example, the body element type name is exposed through the "tagName" property as "BODY".

## 2.5.4. The HTMLElement interface

**Interface *HTMLElement***

All HTML elements derive from this class. Elements that only expose the core attributes are represented by the base `HTMLElement` interface. These elements are as follows:

- HEAD
- special: SUB, SUP, SPAN, BDO
- font: TT, I, B, U, S, STRIKE, BIG, SMALL
- phrase: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ACRONYM, ABBR
- list: DD, DT
- NOFRAMES, NOSCRIPT
- ADDRESS, CENTER

*Note.* The `style` attribute for this interface is reserved for future usage.

**IDL Definition**

```
interface HTMLElement : Element {
           attribute  wstring             id;
           attribute  wstring             title;
           attribute  wstring             lang;
           attribute  wstring             dir;
           attribute  wstring             className;
};
```

**Attributes**

***id***

The element's identifier. See the id attribute definition in HTML 4.0.

***title***

The element's advisory title. See the title attribute definition in HTML 4.0.

***lang***

Language code defined in RFC 1766. See the lang attribute definition in HTML 4.0.

***dir***

Specifies left-to-right or right-to-left direction for weak/neutral text. See the dir attribute definition in HTML 4.0.

***className***

The class attribute of the element. This attribute has been renamed due to conflicts with the "class" keyword exposed by many languages. See the class attribute definition in HTML 4.0.

## 2.5.5. Object definitions

### Interface *HTMLHtmlElement*

Root of an HTML document. See the HTML element definition in HTML 4.0.
**IDL Definition**

```
interface HTMLHtmlElement : HTMLElement {
           attribute  wstring             version;
};
```

**Attributes**

***version***

Version information about the document's DTD. See the version attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLHeadElement*

Document head information. See the HEAD element definition in HTML 4.0.
**IDL Definition**

```
interface HTMLHeadElement : HTMLElement {
           attribute  wstring             profile;
};
```

**Attributes**

***profile***

URI designating a metadata profile. See the profile attribute definition in HTML 4.0.

### Interface *HTMLLinkElement*

The `LINK` element specifies a link to an external resource, and defines this document's relationship to that resource (or vice versa). See the LINK element definition in HTML 4.0.
**IDL Definition**

```
interface HTMLLinkElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  wstring             charset;
           attribute  wstring             href;
           attribute  wstring             hreflang;
           attribute  wstring             media;
           attribute  wstring             rel;
           attribute  wstring             rev;
           attribute  wstring             target;
           attribute  wstring             type;
};
```

**Attributes**

***disabled***

Enables/disables the link. This is currently only used for stylesheet links, and may be used to activate or deactivate stylesheets.

***charset***

The character encoding of the resource being linked to. See the charset attribute definition in HTML 4.0.

***href***

The URI of the linked resource. See the href attribute definition in HTML 4.0.

***hreflang***

Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

***media***

Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

***rel***

Forward link type. See the rel attribute definition in HTML 4.0.

***rev***

Reverse link type. See the rev attribute definition in HTML 4.0.

***target***

Frame to render the resource in. See the target attribute definition in HTML 4.0.

***type***

Advisory content type. See the type attribute definition in HTML 4.0.

### Interface *HTMLTitleElement*

The document title. See the TITLE element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTitleElement : HTMLElement {
           attribute  wstring             text;
};
```

**Attributes**

***text***

The specified title as a string.

### Interface *HTMLMetaElement*

This contains generic meta-information about the document. See the META element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLMetaElement : HTMLElement {
           attribute  wstring             content;
           attribute  wstring             httpEquiv;
           attribute  wstring             name;
           attribute  wstring             scheme;
};
```

**Attributes**

***content***

Associated information. See the content attribute definition in HTML 4.0.

***httpEquiv***

HTTP response header name. See the http-equiv attribute definition in HTML 4.0.

***name***

Meta information name. See the name attribute definition in HTML 4.0.

***scheme***

Select form of content. See the scheme attribute definition in HTML 4.0.

### Interface *HTMLBaseElement*

Document base URI. See the BASE element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLBaseElement : HTMLElement {
           attribute  wstring             href;
           attribute  wstring             target;
};
```

**Attributes**

***href***

The base URI See the href attribute definition in HTML 4.0.

***target***

The default target frame. See the target attribute definition in HTML 4.0.

### Interface *HTMLIsIndexElement*

This element is used for single-line text input. See the ISINDEX element definition in HTML 4.0. This element is deprecated in HTML 4.0.

**IDL Definition**

```
interface HTMLIsIndexElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              prompt;
};
```

**Attributes**

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***prompt***

The prompt message. See the prompt attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLStyleElement*

Style information. A more detailed stylesheet object model will be defined in a separate document. See the STYLE element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLStyleElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  wstring             media;
           attribute  wstring             type;
};
```

**Attributes**

***disabled***

Enables/disables the stylesheet.

***media***

Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

***type***

The style sheet language (Internet media type). See the type attribute definition in HTML 4.0.

### Interface *HTMLBodyElement*

The HTML document body. This element is always present in the DOM API, even if the tags are not present in the source document. See the BODY element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLBodyElement : HTMLElement {
           attribute  wstring             aLink;
           attribute  wstring             background;
           attribute  wstring             bgColor;
           attribute  wstring             link;
           attribute  wstring             text;
           attribute  wstring             vLink;
};
```

**Attributes**

***aLink***

Color of active links (after mouse-button down, but before mouse-button up). See the alink attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***background***

URI of the background texture tile image. See the background attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***bgColor***

Document background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***link***

Color of links that are not active and unvisited. See the link attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***text***

Document text color. See the text attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***vLink***

Color of links that have been visited by the user. See the vlink attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

## Interface *HTMLFormElement*

The `FORM` element encompasses behavior similar to a collection and an element. It provides direct access to the contained input elements as well as the attributes of the form element. See the FORM element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLFormElement : HTMLElement {
  readonly attribute  HTMLCollection       elements;
  readonly attribute  long                 length;
           attribute  wstring              name;
           attribute  wstring              acceptCharset;
           attribute  wstring              action;
           attribute  wstring              enctype;
           attribute  wstring              method;
           attribute  wstring              target;
};
```

**Attributes**

***elements***

Returns a collection of all control elements in the form.

***length***

The number of form controls in the form.

***name***

Names the form.

***acceptCharset***

List of character sets supported by the server. See the accept-charset attribute definition in HTML 4.0.

***action***

Server-side form handler. See the action attribute definition in HTML 4.0.

***enctype***

The content type of the submitted form, generally "application/x-www-form-urlencoded". See the enctype attribute definition in HTML 4.0.

***method***

HTTP method used to submit form. See the method attribute definition in HTML 4.0.

***target***

Frame to render the resource in. See the target attribute definition in HTML 4.0.

### Interface *HTMLSelectElement*

The select element allows the selection of an option. The contained options can be directly accessed through the select element as a collection. See the SELECT element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLSelectElement : HTMLElement {
  readonly attribute  wstring             type;
           attribute  long                selectedIndex;
           attribute  wstring             value;
           attribute  long                length;
  readonly attribute  HTMLFormElement     form;
           attribute  HTMLCollection      options;
           attribute  boolean             disabled;
           attribute  boolean             multiple;
           attribute  wstring             name;
           attribute  long                size;
           attribute  long                tabIndex;
  void                     add(in HTMLElement element,
                               in HTMLElement before);
  void                     remove(in long index);
  void                     blur();
  void                     focus();
};
```

**Attributes**

***type***

The type of control created.

***selectedIndex***

The ordinal index of the selected option. The value -1 is returned if no element is selected. If multiple options are selected, the index of the first selected option is returned.

***value***

The current form control value.

***length***

The number of options in this `SELECT`.

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***options***

The collection of `OPTION` elements contained by this element.

***disabled***

The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

***multiple***

If true, multiple `OPTION` elements may be selected in this `SELECT`. See the multiple attribute definition in HTML 4.0.

***name***

Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

***size***

Number of visible rows. See the size attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

**Methods**

***add***

Add a new element to the collection of `OPTION` elements for this `SELECT`.

**Parameters**

| | |
|---|---|
| **element** | The element to add. |
| **before** | The element to insert before, or NULL for the head of the list. |

This method returns nothing.
This method raises no exceptions.

***remove***

Remove an element from the collection of `OPTION` elements for this `SELECT`. Does nothing if no element has the given index.

**Parameters**

**index** The index of the item to remove.

This method returns nothing.
This method raises no exceptions.

***blur***

Removes keyboard focus from this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***focus***

Gives keyboard focus to this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

## Interface *HTMLOptGroupElement*

Group options together in logical subdivisions. See the OPTGROUP element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLOptGroupElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  wstring             label;
};
```

**Attributes**

***disabled***

The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

***label***

Assigns a label to this option group. See the label attribute definition in HTML 4.0.

### Interface *HTMLOptionElement*

A selectable choice. See the OPTION element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLOptionElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  boolean              defaultSelected;
  readonly attribute  wstring              text;
           attribute  long                 index;
           attribute  boolean              disabled;
           attribute  wstring              label;
  readonly attribute  boolean              selected;
           attribute  wstring              value;
};
```

**Attributes**

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***defaultSelected***

Stores the initial value of the `selected` attribute.

***text***

The text contained within the option element.

***index***

The index of this `OPTION` in its parent `SELECT`.

***disabled***

The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

***label***

Option label for use in hierarchical menus. See the label attribute definition in HTML 4.0.

***selected***

Means that this option is initially selected. See the selected attribute definition in HTML 4.0.

***value***

The current form control value. See the value attribute definition in HTML 4.0.

### Interface *HTMLInputElement*

Form control. *Note.* Depending upon the environment the page is being viewed, the value property may be read-only for the file upload input type. For the "password" input type, the actual value returned may be masked to prevent unauthorized use. See the INPUT element definition in HTML 4.0.

### IDL Definition

```
interface HTMLInputElement : HTMLElement {
           attribute  wstring              defaultValue;
           attribute  boolean              defaultChecked;
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accept;
           attribute  wstring              accessKey;
           attribute  wstring              align;
           attribute  wstring              alt;
           attribute  boolean              checked;
           attribute  boolean              disabled;
           attribute  long                 maxLength;
           attribute  wstring              name;
           attribute  boolean              readOnly;
           attribute  wstring              size;
           attribute  wstring              src;
           attribute  long                 tabIndex;
  readonly attribute  wstring              type;
           attribute  wstring              useMap;
           attribute  wstring              value;
  void                      blur();
  void                      focus();
  void                      select();
  void                      click();
};
```

### Attributes

***defaultValue***

Stores the initial control value (i.e., the initial value of `value`).

***defaultChecked***

When `type` has the value "Radio" or "Checkbox", stores the initial value of the `checked` attribute.

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***accept***

A comma-separated list of content types that a server processing this form will handle correctly. See the accept attribute definition in HTML 4.0.

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***align***

Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***alt***

Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

***checked***

Describes whether a radio or check box is checked, when `type` has the value "Radio" or "Checkbox". The value is TRUE if explicitly set. Represents the current state of the checkbox or radio button. See the checked attribute definition in HTML 4.0.

***disabled***

The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

***maxLength***

Maximum number of characters for text fields, when `type` has the value "Text" or "Password". See the maxlength attribute definition in HTML 4.0.

***name***

Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

***readOnly***

This control is read-only. When `type` has the value "text" or "password" only. See the readonly attribute definition in HTML 4.0.

***size***

Size information. The precise meaning is specific to each type of field. See the size attribute definition in HTML 4.0.

***src***

When the `type` attribute has the value "Image", this attribute specifies the location of the image to be used to decorate the graphical submit button. See the src attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element’s position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

***type***

The type of control created. See the type attribute definition in HTML 4.0.

***useMap***

Use client-side image map. See the usemap attribute definition in HTML 4.0.

***value***

The current form control value. Used for radio buttons and check boxes. See the value attribute definition in HTML 4.0.

**Methods**

***blur***

Removes keyboard focus from this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***focus***

Gives keyboard focus to this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***select***

Select the contents of the text area. For `INPUT` elements whose `type` attribute has one of the following values: "Text", "File", or "Password".

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***click***

Simulate a mouse-click. For `INPUT` elements whose `type` attribute has one of the following values: "Button", "Checkbox", "Radio", "Reset", or "Submit".

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

## Interface *HTMLTextAreaElement*

Multi-line text field. See the TEXTAREA element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTextAreaElement : HTMLElement {
           attribute  wstring             defaultValue;
  readonly attribute  HTMLFormElement     form;
           attribute  wstring             accessKey;
           attribute  long                cols;
           attribute  boolean             disabled;
           attribute  wstring             name;
           attribute  boolean             readOnly;
           attribute  long                rows;
           attribute  long                tabIndex;
  readonly attribute  wstring             type;
```

```
  void                    blur();
  void                    focus();
  void                    select();
};
```

**Attributes**

***defaultValue***

Stores the initial control value (i.e., the initial value of `value`).

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***cols***

Width of control (in characters). See the cols attribute definition in HTML 4.0.

***disabled***

The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

***name***

Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

***readOnly***

This control is read-only. See the readonly attribute definition in HTML 4.0.

***rows***

Number of text rows. See the rows attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

***type***

The type of this form control.

**Methods**

***blur***

Removes keyboard focus from this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***focus***

Gives keyboard focus to this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***select***

Select the contents of the `TEXTAREA`.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

## Interface *HTMLButtonElement*

Push button. See the BUTTON element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLButtonElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accessKey;
           attribute  boolean              disabled;
           attribute  wstring              name;
           attribute  long                 tabIndex;
  readonly attribute  wstring              type;
           attribute  wstring              value;
};
```

**Attributes**

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***disabled***

The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

***name***

Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

***type***

The type of button. See the type attribute definition in HTML 4.0.

***value***

The current form control value. See the value attribute definition in HTML 4.0.

### Interface *HTMLLabelElement*

Form field label text. See the LABEL element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLLabelElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accessKey;
           attribute  wstring              htmlFor;
};
```

**Attributes**

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***htmlFor***

This attribute links this label with another form control by `id` attribute. See the for attribute definition in HTML 4.0.

### Interface *HTMLFieldSetElement*

Organizes form controls into logical groups. See the FIELDSET element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLFieldSetElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
};
```

**Attributes**

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

### Interface *HTMLLegendElement*

Provides a caption for a FIELDSET grouping. See the LEGEND element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLLegendElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accessKey;
           attribute  wstring              align;
};
```

**Attributes**

***form***

Returns the FORM element containing this control. Returns null if this control is not within the context of a form.

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***align***

Text alignment relative to FIELDSET. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLUListElement*

Unordered list. See the UL element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLUListElement : HTMLElement {
           attribute  boolean              compact;
           attribute  wstring              type;
};
```

**Attributes**

***compact***

Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***type***

Bullet style. See the type attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLOListElement*

Ordered list. See the OL element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLOListElement : HTMLElement {
           attribute  boolean             compact;
           attribute  long                start;
           attribute  wstring             type;
};
```

**Attributes**

***compact***

Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***start***

Starting sequence number. See the start attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***type***

Numbering style. See the type attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLDListElement*

Definition list. See the DL element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLDListElement : HTMLElement {
           attribute  boolean             compact;
};
```

**Attributes**

***compact***

Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLDirectoryElement*

Directory list. See the DIR element definition in HTML 4.0. This element is deprecated in HTML 4.0.

**IDL Definition**

```
interface HTMLDirectoryElement : HTMLElement {
           attribute  boolean             compact;
};
```

**Attributes**

***compact***

Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLMenuElement*

Menu list. See the MENU element definition in HTML 4.0. This element is deprecated in HTML 4.0.
**IDL Definition**

```
interface HTMLMenuElement : HTMLElement {
           attribute  boolean             compact;
};
```

**Attributes**

***compact***

Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLLIElement*

List item. See the LI element definition in HTML 4.0.
**IDL Definition**

```
interface HTMLLIElement : HTMLElement {
           attribute  wstring             type;
           attribute  long                value;
};
```

**Attributes**

***type***

List item bullet style. See the type attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***value***

Reset sequence number when used in `OL` See the value attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLBlockquoteElement*

??? See the BLOCKQUOTE element definition in HTML 4.0.
**IDL Definition**

```
interface HTMLBlockquoteElement : HTMLElement {
           attribute  wstring             cite;
};
```

**Attributes**

***cite***

A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

### Interface *HTMLDivElement*

Generic block container. See the DIV element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLDivElement : HTMLElement {
           attribute  wstring             align;
};
```

**Attributes**

***align***

Horizontal text alignment. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLParagraphElement*

Paragraphs. See the P element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLParagraphElement : HTMLElement {
           attribute  wstring             align;
};
```

**Attributes**

***align***

Horizontal text alignment. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLHeadingElement*

For the `H1` to `H6` elements. See the H1 element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLHeadingElement : HTMLElement {
           attribute  wstring             align;
};
```

**Attributes**

***align***

Horizontal text alignment. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLQuoteElement*

For the `Q` and `BLOCKQUOTE` elements. See the Q element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLQuoteElement : HTMLElement {
           attribute  wstring             cite;
};
```

**Attributes**

***cite***

A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

### Interface *HTMLPreElement*

Preformatted text. See the PRE element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLPreElement : HTMLElement {
           attribute  long                width;
};
```

**Attributes**

***width***

Fixed width for content. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLBRElement*

Force a line break. See the BR element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLBRElement : HTMLElement {
           attribute  wstring             clear;
};
```

**Attributes**

***clear***

Control flow of text around floats. See the clear attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLBaseFontElement*

Base font. See the BASEFONT element definition in HTML 4.0. This element is deprecated in HTML 4.0.

**IDL Definition**

```
interface HTMLBaseFontElement : HTMLElement {
           attribute  wstring             color;
           attribute  wstring             face;
           attribute  wstring             size;
};
```

**Attributes**

***color***

Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***face***

Font face identifier. See the face attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***size***

Font size. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLFontElement*

Local change to font. See the FONT element definition in HTML 4.0. This element is deprecated in HTML 4.0.

**IDL Definition**

```
interface HTMLFontElement : HTMLElement {
           attribute  wstring             color;
           attribute  wstring             face;
           attribute  wstring             size;
};
```

**Attributes**

***color***

Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***face***

Font face identifier. See the face attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***size***

Font size. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLHRElement*

Create a horizontal rule. See the HR element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLHRElement : HTMLElement {
           attribute  wstring             align;
           attribute  boolean             noShade;
           attribute  wstring             size;
           attribute  wstring             width;
};
```

**Attributes**

***align***

Align the rule on the page. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***noShade***

Indicates to the user agent that there should be no shading in the rendering of this element. See the noshade attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***size***

The height of the rule. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***width***

The width of the rule. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLInsElement*

Inserted content. See the INS element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLInsElement : HTMLElement {
           attribute  wstring             cite;
           attribute  wstring             dateTime;
};
```

**Attributes**

***cite***

A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

***dateTime***

The date and time of the change. See the datetime attribute definition in HTML 4.0.

### Interface *HTMLDelElement*

Deleted content. See the DEL element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLDelElement : HTMLElement {
           attribute  wstring             cite;
           attribute  wstring             dateTime;
};
```

**Attributes**

***cite***

A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

***dateTime***

The date and time of the change. See the datetime attribute definition in HTML 4.0.

### Interface *HTMLModElement*

Notice of modification to part of a document.

**IDL Definition**

```
interface HTMLModElement : HTMLElement {
           attribute  wstring             cite;
           attribute  wstring             dateTime;
};
```

**Attributes**

***cite***

A URI designating a document that describes the reason for the change. See the HTMLDelElement.

***dateTime***

The date and time of the change. See the HTMLDelElement.

### Interface *HTMLAnchorElement*

The anchor element. See the A element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLAnchorElement : HTMLElement {
           attribute  wstring             accessKey;
           attribute  wstring             charset;
           attribute  wstring             coords;
           attribute  wstring             href;
           attribute  wstring             hreflang;
           attribute  wstring             name;
           attribute  wstring             rel;
           attribute  wstring             rev;
           attribute  wstring             shape;
           attribute  long                tabIndex;
           attribute  wstring             target;
           attribute  wstring             type;
  void                     blur();
  void                     focus();
};
```

**Attributes**

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***charset***

The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

***coords***

Comma-separated list of lengths, defining an active region geometry. See also `shape` for the shape of the region. See the coords attribute definition in HTML 4.0.

***href***

The URI of the linked resource. See the href attribute definition in HTML 4.0.

***hreflang***

Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

***name***

Anchor name. *Note.* For reasons of backwards compatibility, the returned set of anchors only contains those anchors created with the `name` attribute, not those created with the `id` attribute. See the name attribute definition in HTML 4.0.

***rel***

Forward link type. See the rel attribute definition in HTML 4.0.

***rev***

Reverse link type. See the rev attribute definition in HTML 4.0.

***shape***

The shape of the active area. The coordinates are given by `coords`. See the shape attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

***target***

Frame to render the resource in. See the target attribute definition in HTML 4.0.

***type***

Advisory content type. See the type attribute definition in HTML 4.0.

**Methods**

***blur***

Removes keyboard focus from this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***focus***

Gives keyboard focus to this element.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

## Interface *HTMLImageElement*

Embedded image. See the IMG element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLImageElement : HTMLElement {
           attribute  wstring             lowSrc;
           attribute  wstring             name;
           attribute  wstring             align;
           attribute  wstring             alt;
           attribute  wstring             border;
           attribute  wstring             height;
           attribute  wstring             hspace;
           attribute  boolean             isMap;
           attribute  wstring             longDesc;
           attribute  wstring             src;
           attribute  wstring             useMap;
           attribute  wstring             vspace;
           attribute  wstring             width;
};
```

**Attributes**

***lowSrc***

URI designating the source of this image, for low-resolution output.

***name***

The name of the element (for backwards compatibility).

***align***

Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***alt***

Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

***border***

Width of border around image. See the border attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***height***

Override height. See the height attribute definition in HTML 4.0.

***hspace***

Horizontal space to the left and right of this image, applet, or object. See the hspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***isMap***

Use server-side image map. See the ismap attribute definition in HTML 4.0.

***longDesc***

URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

***src***

URI designating the source of this image. See the src attribute definition in HTML 4.0.

***useMap***

Use client-side image map. See the usemap attribute definition in HTML 4.0.

***vspace***

Vertical space above and below this image, applet, or object. See the vspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***width***

Override width. See the width attribute definition in HTML 4.0.

### Interface *HTMLObjectElement*

Generic embedded object. *Note.* In principle, all properties on the object element are read-write but in some environments some properties may be read-only once the underlying object is instantiated. See the OBJECT element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLObjectElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              code;
           attribute  wstring              align;
           attribute  wstring              archive;
           attribute  wstring              border;
           attribute  wstring              codeBase;
           attribute  wstring              codeType;
           attribute  wstring              data;
           attribute  boolean              declare;
           attribute  wstring              height;
           attribute  wstring              hspace;
```

```
            attribute  wstring             name;
            attribute  wstring             standby;
            attribute  long                tabIndex;
            attribute  wstring             type;
            attribute  wstring             useMap;
            attribute  wstring             vspace;
            attribute  wstring             width;
};
```

### Attributes

***form***

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

***code***

Applet class file. See the `code` attribute for HTMLAppletElement.

***align***

Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***archive***

Space-separated list of archives. See the archive attribute definition in HTML 4.0.

***border***

Width of border around the object. See the border attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***codeBase***

Base URI for `classid`, `data`, and `archive` attributes. See the codebase attribute definition in HTML 4.0.

***codeType***

Content type for data downloaded via `classid` attribute. See the codetype attribute definition in HTML 4.0.

***data***

A URI specifying the location of the object's data. See the data attribute definition in HTML 4.0.

***declare***

Declare (for future reference), but do not instantiate, this object. See the declare attribute definition in HTML 4.0.

***height***

Override height. See the height attribute definition in HTML 4.0.

***hspace***

Horizontal space to the left and right of this image, applet, or object. See the hspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***name***

Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

***standby***

Message to render while loading the object. See the standby attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

***type***

Content type for data downloaded via `data` attribute. See the type attribute definition in HTML 4.0.

***useMap***

Use client-side image map. See the usemap attribute definition in HTML 4.0.

***vspace***

Vertical space above and below this image, applet, or object. See the vspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***width***

Override width. See the width attribute definition in HTML 4.0.

## Interface *HTMLParamElement*

Parameters fed to the `OBJECT` element. See the PARAM element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLParamElement : HTMLElement {
           attribute  wstring             name;
           attribute  wstring             type;
           attribute  wstring             value;
           attribute  wstring             valueType;
};
```

**Attributes**

***name***

The name of a run-time parameter. See the name attribute definition in HTML 4.0.

***type***

Content type for the `value` attribute when `valuetype` has the value "ref". See the type attribute definition in HTML 4.0.

***value***

The value of a run-time parameter. See the value attribute definition in HTML 4.0.

***valueType***

Information about the meaning of the `value` attribute value. See the valuetype attribute definition in HTML 4.0.

### Interface *HTMLAppletElement*

Insert a script. See the APPLET element definition in HTML 4.0. This element is deprecated in HTML 4.0.

**IDL Definition**

```
interface HTMLAppletElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             alt;
           attribute  wstring             archive;
           attribute  wstring             code;
           attribute  wstring             codeBase;
           attribute  wstring             height;
           attribute  wstring             hspace;
           attribute  wstring             name;
           attribute  wstring             object;
           attribute  wstring             vspace;
           attribute  wstring             width;
};
```

**Attributes**

***align***

Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***alt***

Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***archive***

Comma-separated archive list. See the archive attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***code***

Applet class file. See the code attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***codeBase***

Optional base URI for applet. See the codebase attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***height***

Override height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***hspace***

Horizontal space to the left and right of this image, applet, or object. See the hspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***name***

The name of the applet. See the name attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***object***

Serialized applet file. See the object attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***vspace***

Vertical space above and below this image, applet, or object. See the vspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***width***

Override width. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLMapElement*

Client-side image map. See the MAP element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLMapElement : HTMLElement {
  readonly attribute  HTMLCollection       areas;
           attribute  wstring              name;
};
```

**Attributes**

***areas***

The list of areas defined for the image map.

***name***

Names the map (for use with `usemap`). See the name attribute definition in HTML 4.0.

### Interface *HTMLAreaElement*

Client-side image map area definition. See the AREA element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLAreaElement : HTMLElement {
           attribute  wstring             accessKey;
           attribute  wstring             alt;
           attribute  wstring             coords;
           attribute  wstring             href;
           attribute  boolean             noHref;
           attribute  wstring             shape;
           attribute  long                tabIndex;
           attribute  wstring             target;
};
```

**Attributes**

***accessKey***

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

***alt***

Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

***coords***

Comma-separated list of lengths, defining an active region geometry. See also `shape` for the shape of the region. See the coords attribute definition in HTML 4.0.

***href***

The URI of the linked resource. See the href attribute definition in HTML 4.0.

***noHref***

Specifies that this area is inactive, i.e., has no associated action. See the nohref attribute definition in HTML 4.0.

***shape***

The shape of the active area. The coordinates are given by `coords`. See the shape attribute definition in HTML 4.0.

***tabIndex***

Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

***target***

Frame to render the resource in. See the target attribute definition in HTML 4.0.

**Interface *HTMLScriptElement***

Script statements. See the SCRIPT element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLScriptElement : HTMLElement {
           attribute  wstring             text;
           attribute  wstring             htmlFor;
           attribute  wstring             event;
           attribute  wstring             charset;
           attribute  boolean             defer;
           attribute  wstring             src;
           attribute  wstring             type;
};
```

**Attributes**

***text***

The script content of the element.

***htmlFor***

*Reserved for future use.*

***event***

*Reserved for future use.*

***charset***

The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

***defer***

Indicates that the user agent can defer processing of the script. See the defer attribute definition in HTML 4.0.

***src***

URI designating an external script. See the src attribute definition in HTML 4.0.

***type***

The content type of the script language. See the type attribute definition in HTML 4.0.

### Interface *HTMLTableElement*

The create* and delete* methods on the table allow authors to construct and modify tables. HTML 4.0 specifies that a table may only contain one `CAPTION`, `THEAD`, and `TFOOT` element can exist in a table. Therefore, one exists, and the createTHead() or createTFoot() method is called, the method returns the existing THead or TFoot element. See the TABLE element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTableElement : HTMLElement {
           attribute  HTMLTableCaptionElement caption;
           attribute  HTMLTableSectionElement tHead;
           attribute  HTMLTableSectionElement tFoot;
  readonly attribute  HTMLCollection       rows;
           attribute  HTMLCollection       tBodies;
           attribute  wstring              align;
           attribute  wstring              bgColor;
           attribute  wstring              border;
```

```
           attribute  wstring             cellPadding;
           attribute  wstring             cellSpacing;
           attribute  wstring             frame;
           attribute  wstring             rules;
           attribute  wstring             summary;
           attribute  wstring             width;
  HTMLElement              createTHead();
  void                     deleteTHead();
  HTMLElement              createTFoot();
  void                     deleteTFoot();
  HTMLElement              createCaption();
  void                     deleteCaption();
  HTMLElement              insertRow(in long index);
  void                     deleteRow(in long index);
};
```

### Attributes

***caption***

Returns the table's `CAPTION`, or void if none exists.

***tHead***

Returns the table's `THEAD`, or void if none exists.

***tFoot***

Returns the table's `TFOOT`, or void if none exists.

***rows***

Returns a collection of all rows of the table.

***tBodies***

Returns a collection of the defined table bodies.

***align***

Specifies the table's position with respect to the rest of the document. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***bgColor***

Cell background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***border***

The width of the border around the table. See the border attribute definition in HTML 4.0.

***cellPadding***

Specifies the horizontal and vertical space between cell content and cell borders. See the cellpadding attribute definition in HTML 4.0.

***cellSpacing***

Specifies the horizontal and vertical separation between cells. See the cellspacing attribute definition in HTML 4.0.

***frame***

Specifies which external table borders to render. See the frame attribute definition in HTML 4.0.

***rules***

Specifies which internal table borders to render. See the rules attribute definition in HTML 4.0.

***summary***

Supplementary description about the purpose or structure of a table. See the summary attribute definition in HTML 4.0.

***width***

Specifies the desired table width. See the width attribute definition in HTML 4.0.

**Methods**

***createTHead***

Create a table header row or return an existing one.

**Return Values**

A new table header element (`THEAD`).

This method has no parameters.

This method raises no exceptions.

***deleteTHead***

Delete the header from the table, if one exists.

This method has no parameters.

This method returns nothing.

This method raises no exceptions.

***createTFoot***

Create a table footer row or return an existing one.

**Return Values**

A footer element (`TFOOT`).

This method has no parameters.

This method raises no exceptions.

***deleteTFoot***

Delete the footer from the table, if one exists.

This method has no parameters.

This method returns nothing.

This method raises no exceptions.

***createCaption***

Create a new table caption object or return an existing one.
**Return Values**

A `CAPTION` element.
This method has no parameters.
This method raises no exceptions.

***deleteCaption***

Delete the table caption, if one exists.

This method has no parameters.
This method returns nothing.
This method raises no exceptions.

***insertRow***

Insert a new empty row in the table. *Note.* A table row cannot be empty according to HTML 4.0 Recommendation.
**Parameters**

**index** The row number where to insert a new row.

**Return Values**

The newly created row.
This method raises no exceptions.

***deleteRow***

Delete a table row.
**Parameters**

**index** The index of the row to be deleted.

This method returns nothing.
This method raises no exceptions.

## Interface *HTMLTableCaptionElement*

Table caption See the CAPTION element definition in HTML 4.0.
**IDL Definition**

```
interface HTMLTableCaptionElement : HTMLElement {
           attribute  wstring             align;
};
```

**Attributes**

***align***

Caption alignment with respect to the table. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLTableColElement*

Regroups the `COL` and `COLGROUP` elements. See the COL element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTableColElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  long                span;
           attribute  wstring             vAlign;
           attribute  wstring             width;
};
```

**Attributes**

***align***

Horizontal alignment of cell data in column. See the align attribute definition in HTML 4.0.

***ch***

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

***chOff***

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

***span***

Indicates the number of columns in a group or affected by a grouping. See the span attribute definition in HTML 4.0.

***vAlign***

Vertical alignment of cell data in column. See the valign attribute definition in HTML 4.0.

***width***

Default column width. See the width attribute definition in HTML 4.0.

### Interface *HTMLTheadElement*

`THEAD` element. See the THEAD element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTheadElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
};
```

**Attributes**

***align***

Horizontal alignment of data in table header cells. See the align attribute definition in HTML 4.0.

***ch***

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

***chOff***

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

***vAlign***

Vertical alignment of data in table header cells. See the valign attribute definition in HTML 4.0.

## Interface *HTMLTbodyElement*

`TBODY` See the TBODY element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTbodyElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
};
```

**Attributes**

***align***

Horizontal alignment of data in table footer cells. See the align attribute definition in HTML 4.0.

***ch***

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

***chOff***

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

***vAlign***

Vertical alignment of data in table footer cells. See the valign attribute definition in HTML 4.0.

### Interface *HTMLTfootElement*

`TFOOT` element. See the TFOOT element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTfootElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
};
```

**Attributes**

***align***

Specifies horizontal alignment. See the align attribute definition in HTML 4.0.

***ch***

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

***chOff***

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

***vAlign***

Vertical alignment. See the valign attribute definition in HTML 4.0.

### Interface *HTMLTableSectionElement*

The `THEAD`, `TFOOT`, and `TBODY` elements.

**IDL Definition**

```
interface HTMLTableSectionElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             vAlign;
           attribute  HTMLCollection      rows;
  HTMLElement              insertRow(in long index);
  void                     deleteRow(in long index);
};
```

**Attributes**

***align***

Horizontal alignment of data in cells. See the `align` attribute for HTMLTheadElement for details.

***vAlign***

Vertical alignment of data in cells. See the `valign` attribute for HTMLTheadElement for details.

***rows***

The collection of rows in this table section.

**Methods**

***insertRow***

Insert a row into this section.

**Parameters**

**index** The row number where to insert a new row.

**Return Values**

The newly created row.

This method raises no exceptions.

***deleteRow***

Delete a row from this section.

**Parameters**

**index** The index of the row to be deleted.

This method returns nothing.

This method raises no exceptions.

## Interface *HTMLTableRowElement*

A row in a table. See the TR element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTableRowElement : HTMLElement {
           attribute  long                rowIndex;
           attribute  long                sectionRowIndex;
           attribute  HTMLCollection      cells;
           attribute  wstring             align;
           attribute  wstring             bgColor;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
  HTMLElement              insertCell(in long index);
  void                     deleteCell(in long index);
};
```

**Attributes**

***rowIndex***

The index of this row, relative to the entire table.

***sectionRowIndex***

The index of this row, relative to the current section (`THEAD`, `TFOOT`, or `TBODY`).

***cells***

The collection of cells in this row.

***align***

Horizontal alignment of data within cells of this row. See the align attribute definition in HTML 4.0.

***bgColor***

Background color for rows. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***ch***

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

***chOff***

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

***vAlign***

Vertical alignment of data within cells of this row. See the valign attribute definition in HTML 4.0.

**Methods**

***insertCell***

Insert an empty cell into this row.

**Parameters**

**index** The place to insert the cell.

**Return Values**

The newly created cell.

This method raises no exceptions.

***deleteCell***

Delete a cell from the current row.

**Parameters**

**index** The index of the cell to delete.

This method returns nothing.

This method raises no exceptions.

**Interface *HTMLTableCellElement***

The object used to represent the `TH` and `TD` elements. See the TD element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLTableCellElement : HTMLElement {
           attribute  long                cellIndex;
           attribute  wstring             abbr;
           attribute  wstring             align;
           attribute  wstring             axis;
           attribute  wstring             bgColor;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  long                colSpan;
           attribute  wstring             headers;
           attribute  wstring             height;
           attribute  boolean             noWrap;
           attribute  long                rowSpan;
           attribute  wstring             scope;
           attribute  wstring             vAlign;
           attribute  wstring             width;
};
```

**Attributes**

***cellIndex***

The index of this cell in the row.

***abbr***

Abbreviation for header cells. See the abbr attribute definition in HTML 4.0.

***align***

Horizontal alignment of data in cell. See the align attribute definition in HTML 4.0.

***axis***

Names group of related headers. See the axis attribute definition in HTML 4.0.

***bgColor***

Cell background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***ch***

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

***chOff***

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

***colSpan***

Number of columns spanned by cell. See the colspan attribute definition in HTML 4.0.

***headers***

List of `id` attribute values for header cells. See the headers attribute definition in HTML 4.0.

***height***

Cell height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***noWrap***

Suppress word wrapping. See the nowrap attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***rowSpan***

Number of rows spanned by cell. See the rowspan attribute definition in HTML 4.0.

***scope***

Scope covered by header cells. See the scope attribute definition in HTML 4.0.

***vAlign***

Vertical alignment of data in cell. See the valign attribute definition in HTML 4.0.

***width***

Cell width. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

### Interface *HTMLFrameSetElement*

Create a grid of frames. See the FRAMESET element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLFrameSetElement : HTMLElement {
           attribute  wstring             cols;
           attribute  wstring             rows;
};
```

**Attributes**

***cols***

The number of columns of frames in the frameset. See the cols attribute definition in HTML 4.0.

***rows***

The number of rows of frames in the frameset. See the rows attribute definition in HTML 4.0.

### Interface *HTMLFrameElement*

Create a frame. See the FRAME element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLFrameElement : HTMLElement {
           attribute  wstring             frameBorder;
           attribute  wstring             longDesc;
           attribute  wstring             marginHeight;
           attribute  wstring             marginWidth;
           attribute  wstring             name;
           attribute  boolean             noResize;
           attribute  wstring             scrolling;
           attribute  wstring             src;
};
```

**Attributes**

***frameBorder***

Request frame borders. See the frameborder attribute definition in HTML 4.0.

***longDesc***

URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

***marginHeight***

Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

***marginWidth***

Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

***name***

The frame name (object of the `target` attribute). See the name attribute definition in HTML 4.0.

***noResize***

When true, forbid user from resizing frame. See the noresize attribute definition in HTML 4.0.

***scrolling***

Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

***src***

A URI designating the initial frame contents. See the src attribute definition in HTML 4.0.

## Interface *HTMLIFrameElement*

Inline subwindows. See the IFRAME element definition in HTML 4.0.

**IDL Definition**

```
interface HTMLIFrameElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             frameBorder;
           attribute  wstring             height;
           attribute  wstring             longDesc;
           attribute  wstring             marginHeight;
```

```
            attribute  wstring             marginWidth;
            attribute  wstring             name;
            attribute  wstring             scrolling;
            attribute  wstring             src;
            attribute  wstring             width;
};
```

### Attributes

***align***

Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

***frameBorder***

Request frame borders. See the frameborder attribute definition in HTML 4.0.

***height***

Frame height. See the height attribute definition in HTML 4.0.

***longDesc***

URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

***marginHeight***

Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

***marginWidth***

Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

***name***

The frame name (object of the `target` attribute). See the name attribute definition in HTML 4.0.

***scrolling***

Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

***src***

A URI designating the initial frame contents. See the src attribute definition in HTML 4.0.

***width***

Frame width. See the width attribute definition in HTML 4.0.

# Appendix A: Production Notes

*Editors*

Gavin Nicol, Inso EPS

The DOM specification serves as a good example of the power of using XML: all of the HTML documents, Java bindings, OMG IDL bindings, and ECMA Script bindings are generated froma single set of XML source files. This section outlines how this specification is written in XML, and how the various derived works are created.

# A.1: The Document Type Definition

This specification was written entirely in XML, using a DTD based heavily on the DTD used by the XML Working Group for the XML specification. The major difference between the DTD used by the XML Working Group, and the DTD used for this specification is in the addition of a DTD module for interface specifications.

The DTD module for interfaces specifications is a very loose translation of the Extended Backus-Naur Form (EBNF) specification of the OMG IDL syntax into XML DTD syntax. In addition to the translation, the ability to *describe* the interfaces was added, thereby creating a limited form of *literate programming* for interface definitions.

While the DTD module is sufficient for the purposes of the DOM WG, it is very loosely typed, meaning that there are very few constraints placed on the type specifications (the type information is effectively treated as an opaque string). In a DTD for object to object communication, some stricter enforcement of data types would probably be beneficial.

# A.2: The production process

The DOM specification is written using XML. All documents are valid XML. In order to produce the HTML versions of the specification, the object indexes, the Java source code, and the OMG IDL and ECMA Script definitions, the XML specification is *converted*.

The tool currently used for conversion is *COST* by Joe English. `COST` takes the ESIS output of `nsgmls`, creates an internal representation, and then allows *scripts*, and *event handlers* to be run over the internal data structure. Event handlers allow document *patterns* and associated processing to be specified: when the pattern is matched during a preorder traversal of a document subtree, the associated action is executed. This is the heart of the conversion process. Scripts are used to tie the various components together. For example, each of the major derived data sources (Java code etc.) is created by the execution of a script, which in turn executes one or more event handlers. The scripts and event handlers are specified using TCL.

The current version of `COST` has been somewhat modified from the publically available version. In particular, it now runs correctly under 32-bit Windows, uses TCL 8.0, and correctly handles the case sensitivity of XML (though it probably could not correctly handle native language markup).

We could also have used `Jade`, by James Clark. Like `COST`, `Jade` allows patterns and actions to be specified, but `Jade` is based on DSSSL, an international standard, whereas `COST` is not. `Jade` is more powerful than `COST` in many ways, but prior experience of the editor with Cost made it easier to use this rather than `Jade`. A future version or Level of the DOM specification may be produced using `Jade` or an `XSL` processor.

# A.3: Object Definitions

As stated earlier, all object definitions are specified in XML. The Java bindings, OMG IDL bindings, and ECMA Script bindings are all generated automatically from the XML source code.

This is possible because the information specified in XML is a *superset* of what these other syntax need. This is a general observation, and the same kind of technique can be applied to many other areas: given rich structure, rich processing and conversion are possible. For Java and OMG IDL, it is basically just a matter of renaming syntactic keywords, for ECMA Script, the process is somewhat more involved.

A typical object definition in XML looks something like this:

```
<interface name="foo">
  <descr><p>Description goes here...</p></descr>
  <method name="bar">
    <descr><p>Description goes here...</p></descr>
    <parameters>
      <param name="baz" type="wstring" attr="in">
        <descr><p>Description goes here...</p></descr>
      </param>
    </parameters>
    <returns type="void">
       <descr><p>Description goes here...</p></descr>
    </returns>
    <raises>
       <!-- Throws no exceptions -->
    </raises>
  </method>
</interface>
```

As can easily be seen, this is quite verbose, but not unlike OMG IDL. In fact, when the specification was originally converted to use XML, the OMG IDL definitions were automatically converted into the corresponding XML source using common Unix text manipulation tools.

# Appendix B: Document Object Model Contributors

***Note:*** *This specification was prepared and approved for publication by the W3C DOM Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval.*

Members of the DOM Activity contributing to this specification were:

*Editors*

Lauren Wood, SoftQuad, Inc., *chair*
Arnaud Le Hors, W3C, *W3C staff contact*
Andrew Watson, Object Management Group
Bill Smith, Sun
Chris Lovett, Microsoft
Chris Wilson, Microsoft
David Brownell, Sun
David Singer, IBM
Don Park, invited
Eric Vasilik, Microsoft
Gavin Nicol, INSO
Ian Jacobs, W3C
James Clark, invited
Jared Sorensen, Novell
Jonathan Robie, Texcel
Mike Champion, ArborText
Paul Grosso, ArborText
Peter Sharpe, SoftQuad, Inc.
Phil Karlton, Netscape
Ray Whitmer, iMall
Rich Rollman, Microsoft
Rick Gessner, Netscape
Robert Sutor, IBM
Scott Isaacs, Microsoft
Sharon Adler, INSO
Steve Byrne, JavaSoft
Tim Bray, invited
Tom Pixley, Netscape
Vidur Apparao, Netscape

Appendix B: Document Object Model Contributors

# Appendix C: Document Object Model Glossary

*Editors*
Robert S. Sutor, IBM

Several of the following term definitions have been borrowed or modified from similar definitions in other W3C documents. Links to these documents follow the definitions.

**ancestor**
An *ancestor* node of any node A is any node above A in a tree model of a document, where "above" means "toward the root."

**API**
An *API* is an application programming interface, a set of functions or methods used to access some functionality.

**child**
A *child* is an immediate descendant node of a node.

**client application**
A [client] application is any software that uses the Document Object Model programming interfaces provided by the hosting implementation to accomplish useful work. Some examples of client applications are scripts within an HTML or XML document.

**COM**
*COM* is Microsoft's Component Object Model, a technology for building applications from binary software components.

**content model**
The *content model* is a simple grammar governing the allowed types of the child elements and the order in which they appear. See [XML ]

**context**
A *context* specifies an access pattern (or path): a set of interfaces which give you a way to interact with a model. For example, imagine a model with different colored arcs connecting data nodes. A context might be a sheet of colored acetate that is placed over the model allowing you a partial view of the total information in the model.

**convenience**
A *convenience method* is an operation on an object that could be accomplished by a program consisting of more basic operations on the object. Convenience methods are usually provided to make the API easier and simpler to use or to allow specific programs to create more optimized implementations for common operations. A similar definition holds for a *convenience property*.

**cooked model**
A model for a document that represents the document after it has been manipulated in some way. For example, any combination of any of the following transformations would create a cooked model:

1. Expansion of internal text entities.
2. Expansion of external entities.
3. Model augmentation with style-specified generated text.
4. Execution of style-specified reordering.
5. Execution of scripts.

A browser might only be able to provide access to a cooked model, while an editor might provide access to a cooked or the initial structure model (also known as the *uncooked model*) for a document.

**CORBA**
*CORBA* is the *Common Object Request Broker Architecture* from the OMG. This architecture is a collection of objects and libraries that allow the creation of applications containing objects that make and receive requests and responses in a distributed environment.

**cursor**
A *cursor* is an object representation of a node. It may possess information about context and the path traversed to reach the node.

**data model**
A *data model* is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them.

**deprecation**
When new releases of specifications are released, some older features may be marked as being *deprecated*. This means that new work should not use the features and that although they are supported in the current release, they may not be supported or available in future releases.

**descendant**
A *descendant* node of any node A is any node below A in a tree model of a document, where "above" means "toward the root."

**ECMAScript**
The programming language defined by the ECMA-262 standard. As stated in the standard, the originating technology for ECMAScript was JavaScript. Note that in the ECMAScript binding, the word "property" is used in the same sense as the IDL term "attribute."

**element**
Each document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements by an empty-element tag. Each element has a type, identified by name, and may have a set of attributes. Each attribute has a name and a value. [XML ]

**event propagation, also known as event bubbling**
This is the idea that an event can affect one object and a set of related objects. Any of the potentially affected objects can block the event or substitute a different one (upward event propagation). The event is broadcast from the node at which it originates to every parent node.

**hosting implementation**
A [hosting] implementation is a software module that provides an implementation of the DOM interfaces so that a client application can use them. Some examples of hosting implementations are browsers, editors and document repositories.

**HTML**
The HyperText Markup Language (*HTML*) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. [HTML 3.2 ] [HTML4.0 ]

**IDL**
An Interface Definition Language (*IDL*) is used to define the interfaces for accessing and operating upon objects. Examples of IDLs are the Object Management Group’s IDL , Microsoft’s IDL , and Sun’s Java IDL .

**implementor**
Companies, organizations, and individuals that claim to support the Document Object Model as an API for their products.

**inheritance**
In object-oriented programming, the ability to create new classes (or interfaces) that contain all the methods and properties of another class (or interface), plus additional methods and properties. If class (or interface) D inherits from class (or interface) B, then D is said to be *derived* from B. B is said to be a *base* class (or interface) for D. Some programming languages allow for multiple inheritance, that

is, inheritance from more than one class or interface.

**initial structure model**

Also known as the *raw structure model* or the *uncooked model*, this represents the document before it has been modified by entity expansions, generated text, style-specified reordering, or the execution of scripts. In some implementations, this might correspond to the "initial parse tree" for the document, if it ever exists. Note that a given implementation might not be able to provide access to the initial structure model for a document, though an editor probably would.

**interface**

An *interface* is a declaration of a set of methods with no information given about their implementation. In object systems that support interfaces and inheritance, interfaces can usually inherit from one another.

**language binding**

A programming *language binding* for an IDL specification is an implementation of the interfaces in the specification for the given language. For example, a Java language binding for the Document Object Model IDL specification would implement the concrete Java classes that provide the functionality exposed by the interfaces.

**method**

A *method* is an operation or function that is associated with an object and is allowed to manipulate the object's data.

**model**

A *model* is the actual data representation for the information at hand. Examples are the structural model and the style model representing the parse structure and the style information associated with a document. The model might be a tree, or a directed graph, or something else.

**object model**

An *object model* is a collection of descriptions of classes or interfaces, together with their member data, member functions, and class-static operations.

**parent**

A *parent* is an immediate ancestor node of a node.

**root node**

The *root node* is the unique node that is not a child of any other node. All other nodes are children or other descendents of the root node. [XML ]

**sibling**

Two nodes are *siblings* if they have the same parent node.

**string comparison**

When string matching is required, it is to occur as though the comparison was between 2 sequences of code points from the Unicode 2.0 standard.

**tag valid document**

A document is *tag valid* if all begin and end tags are properly balanced and nested.

**type valid document**

A document is *type valid* if it conforms to an explicit DTD.

**uncooked model**

See initial structure model.

**well-formed document**

A document is *well-formed* if it is tag valid and entities are limited to single elements (i.e., single sub-trees).

**XML**

Extensible Markup Language (*XML*) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. [XML ]

# Appendix D: Document Object Model Level 1 Bindings

This appendix contains the complete OMG IDL, Java, and ECMA Script bindings for the Level 1 Document Object Model definitions. The definitions are divided into Core and HTML.

The IDL and Java bindings are also available as idlbindings.zip and javabindings.zip.

# D.1: Document Object Model Level 1 Core Bindings

## D.1.1: OMG IDL Definitions for Level 1 Core

This section contains the OMG IDL definitions for the interfaces in the Core Document Object Model specification, including the extended (XML) interfaces. The HTML OMG IDL definition is in the Level 1 HTML DOM specification [p.41] .

```
enum ExceptionCode {
  UNSUPPORTED_DOCUMENT_ERR,
  NOT_CHILD_ERR,
  NO_CHILDREN_ALLOWED_ERR,
  INDEX_SIZE_ERR,
  WSTRING_SIZE_ERR,
  DATA_SIZE_ERR
};

exception DOMException {
   ExceptionCode   code;
};

interface DOMImplementation {
  boolean                   hasFeature(in wstring feature,
                                       in wstring version);
};

interface DocumentFragment : Node {
  readonly attribute  Document             masterDoc;
};

interface Document : DocumentFragment {
  readonly attribute  DocumentType         doctype;
  readonly attribute  DOMImplementation    implementation;
  readonly attribute  Element              documentElement;
  Element                   createElement(in wstring tagName);
  DocumentFragment          createDocumentFragment();
  Text                      createTextNode(in wstring data);
  Comment                   createComment(in wstring data);
  CDATASection              createCDATASection(in wstring data);
  ProcessingInstruction     createProcessingInstruction(in wstring target,
                                                        in wstring data);
  Attribute                 createAttribute(in wstring name);
  Entity                    createEntity();
  EntityReference           createEntityReference();
  NodeList                  getElementsByTagName(in wstring tagname);
```

```
};

interface Node {
  // NodeType
  const unsigned short      DOCUMENT           = 1;
  const unsigned short      ELEMENT            = 2;
  const unsigned short      ATTRIBUTE          = 3;
  const unsigned short      PROCESSING_INSTRUCTION = 4;
  const unsigned short      COMMENT            = 5;
  const unsigned short      TEXT               = 6;
  const unsigned short      CDATA_SECTION      = 7;
  const unsigned short      DOCUMENT_FRAGMENT  = 8;
  const unsigned short      ENTITY             = 9;
  const unsigned short      ENTITY_REFERENCE   = 10;
  const unsigned short      DOCUMENT_TYPE      = 11;

  readonly attribute  wstring             nodeName;
           attribute  wstring             nodeValue;
  readonly attribute  unsigned short      nodeType;
  readonly attribute  Node                parentNode;
  readonly attribute  NodeList            childNodes;
  readonly attribute  Node                firstChild;
  readonly attribute  Node                lastChild;
  readonly attribute  Node                previousSibling;
  readonly attribute  Node                nextSibling;
  readonly attribute  NamedNodeMap        attributes;
  Node                      insertBefore(in Node newChild,
                                         in Node refChild)
                                         raises(DOMException);
  Node                      replaceChild(in Node newChild,
                                         in Node oldChild)
                                         raises(DOMException);
  Node                      removeChild(in Node oldChild)
                                       raises(DOMException);
  Node                      appendChild(in Node newChild);
  boolean                   hasChildNodes();
  Node                      cloneNode(in boolean deep);
  boolean                   equals(in Node arg,
                                   in boolean deep);
};

interface NodeList {
  Node                      item(in unsigned long index);
  readonly attribute  unsigned long       size;
};

interface NamedNodeMap {
  Node                      getNamedItem(in wstring name);
  void                      setNamedItem(in Node arg);
  Node                      removeNamedItem(in wstring name);
  Node                      item(in unsigned long index);
  readonly attribute  unsigned long       size;
};

interface Data : Node {
           attribute  wstring             data;
  readonly attribute  unsigned long       size;
```

```
  wstring                    substring(in unsigned long start,
                                       in unsigned long count)
                                       raises(DOMException);
  void                       append(in wstring arg);
  void                       insert(in unsigned long offset,
                                    in wstring arg)
                                    raises(DOMException);
  void                       delete(in unsigned long offset,
                                    in unsigned long count)
                                    raises(DOMException);
  void                       replace(in unsigned long offset,
                                     in unsigned long count,
                                     in wstring arg)
                                     raises(DOMException);
};

interface Attribute : Node {
  wstring                    getName();
           attribute  boolean              specified;
  wstring                    getValue();
};

interface Element : Node {
  wstring                    getTagName();
  NamedNodeMap               getAttributes();
  wstring                    getAttribute(in wstring name);
  void                       setAttribute(in string name,
                                          in string value);
  void                       removeAttribute(in wstring name);
  Attribute                  getAttributeNode(in wstring name);
  void                       setAttributeNode(in Attribute newAttr);
  void                       removeAttributeNode(in Attribute oldAttr);
  NodeList                   getElementsByTagName(in wstring tagname);
  void                       normalize();
};

interface Text : Data {
  Text                       splitText(in unsigned long offset);
  Text                       joinText(in Text node1,
                                      in Text node2);
};

interface Comment : Data {
};

interface ProcessingInstruction : Node {
           attribute  wstring              target;
           attribute  wstring              data;
};

interface CDATASection : Text {
};

interface DocumentType : Node {
           attribute  wstring              name;
  readonly attribute  NamedNodeMap         entities;
  readonly attribute  NamedNodeMap         notations;
```

```
};

interface Notation : Node {
           attribute  wstring             publicId;
           attribute  wstring             systemId;
};

interface Entity : Node {
           attribute  wstring             publicId;
           attribute  wstring             systemId;
           attribute  wstring             notationName;
};

interface EntityReference : Node {
};
```

# D.1.2: Java Definitions for Level 1 Core

```
 public static final int UNSUPPORTED_DOCUMENT_ERR= 1;
 public static final int NOT_CHILD_ERR= 2;
 public static final int NO_CHILDREN_ALLOWED_ERR= 3;
 public static final int INDEX_SIZE_ERR= 4;
 public static final int WSTRING_SIZE_ERR= 5;
 public static final int DATA_SIZE_ERR= 6;


public class DOMException extends RuntimeException {
 public static final int UNSUPPORTED_DOCUMENT_ERR= 1;
 public static final int NOT_CHILD_ERR= 2;
 public static final int NO_CHILDREN_ALLOWED_ERR= 3;
 public static final int INDEX_SIZE_ERR= 4;
 public static final int WSTRING_SIZE_ERR= 5;
 public static final int DATA_SIZE_ERR= 6;


   public int   code;
}

public interface DOMImplementation {
   public boolean            hasFeature(String feature,
                                        String version);
}

public interface DocumentFragment extends Node {
   public Document           getMasterDoc();

}

public interface Document extends DocumentFragment {
   public DocumentType       getDoctype();

   public DOMImplementation  getImplementation();

   public Element            getDocumentElement();

   public Element            createElement(String tagName);
```

```
   public DocumentFragment   createDocumentFragment();
   public Text               createTextNode(String data);
   public Comment            createComment(String data);
   public CDATASection       createCDATASection(String data);
   public ProcessingInstruction createProcessingInstruction(String target,
                                                             String data);
   public Attribute          createAttribute(String name);
   public Entity             createEntity();
   public EntityReference    createEntityReference();
   public NodeList           getElementsByTagName(String tagname);
}

public interface Node {
   // NodeType
   public static final short           DOCUMENT             = 1;
   public static final short           ELEMENT              = 2;
   public static final short           ATTRIBUTE            = 3;
   public static final short           PROCESSING_INSTRUCTION = 4;
   public static final short           COMMENT              = 5;
   public static final short           TEXT                 = 6;
   public static final short           CDATA_SECTION        = 7;
   public static final short           DOCUMENT_FRAGMENT    = 8;
   public static final short           ENTITY               = 9;
   public static final short           ENTITY_REFERENCE     = 10;
   public static final short           DOCUMENT_TYPE        = 11;

   public String             getNodeName();

   public String             getNodeValue();
   public void               setNodeValue(String arg);

   public short              getNodeType();

   public Node               getParentNode();

   public NodeList           getChildNodes();

   public Node               getFirstChild();

   public Node               getLastChild();

   public Node               getPreviousSibling();

   public Node               getNextSibling();

   public NamedNodeMap       getAttributes();

   public Node               insertBefore(Node newChild,
                                          Node refChild)
                                          throws DOMException;
   public Node               replaceChild(Node newChild,
                                          Node oldChild)
                                          throws DOMException;
   public Node               removeChild(Node oldChild)
                                         throws DOMException;
   public Node               appendChild(Node newChild);
   public boolean            hasChildNodes();
```

```
   public Node               cloneNode(boolean deep);
   public boolean            equals(Node arg,
                                    boolean deep);
}

public interface NodeList {
   public Node               item(int index);
   public int                getSize();

}

public interface NamedNodeMap {
   public Node               getNamedItem(String name);
   public void               setNamedItem(Node arg);
   public Node               removeNamedItem(String name);
   public Node               item(int index);
   public int                getSize();

}

public interface Data extends Node {
   public String             getData();
   public void               setData(String arg);

   public int                getSize();

   public String             substring(int start,
                                       int count)
                                       throws DOMException;
   public void               append(String arg);
   public void               insert(int offset,
                                    String arg)
                                    throws DOMException;
   public void               delete(int offset,
                                    int count)
                                    throws DOMException;
   public void               replace(int offset,
                                     int count,
                                     String arg)
                                     throws DOMException;
}

public interface Attribute extends Node {
   public String             getName();
   public boolean            getSpecified();
   public void               setSpecified(boolean arg);

   public String             getValue();
}

public interface Element extends Node {
   public String             getTagName();
   public NamedNodeMap       getAttributes();
   public String             getAttribute(String name);
   public void               setAttribute(String name,
                                          String value);
   public void               removeAttribute(String name);
```

```
   public Attribute          getAttributeNode(String name);
   public void               setAttributeNode(Attribute newAttr);
   public void               removeAttributeNode(Attribute oldAttr);
   public NodeList           getElementsByTagName(String tagname);
   public void               normalize();
}

public interface Text extends Data {
   public Text               splitText(int offset);
   public Text               joinText(Text node1,
                                      Text node2);
}

public interface Comment extends Data {
}

public interface ProcessingInstruction extends Node {
   public String             getTarget();
   public void               setTarget(String arg);

   public String             getData();
   public void               setData(String arg);

}

public interface CDATASection extends Text {
}

public interface DocumentType extends Node {
   public String             getName();
   public void               setName(String arg);

   public NamedNodeMap       getEntities();

   public NamedNodeMap       getNotations();

}

public interface Notation extends Node {
   public String             getPublicId();
   public void               setPublicId(String arg);

   public String             getSystemId();
   public void               setSystemId(String arg);

}

public interface Entity extends Node {
   public String             getPublicId();
   public void               setPublicId(String arg);

   public String             getSystemId();
   public void               setSystemId(String arg);

   public String             getNotationName();
   public void               setNotationName(String arg);
```

```
}

public interface EntityReference extends Node {
}
```

## D.1.3: ECMA Script Definitions for Level 1 Core

# Object ExceptionCode

# Object DOMException

# Object DOMImplementation

The **DOMImplementation** object has the following methods:

**hasFeature(feature, version)**
: This method returns a **boolean**. The **feature** parameter is expected to be of type **wstring.**. The **version** parameter is expected to be of type **wstring.**.

# Object DocumentFragment

**DocumentFragment** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **DocumentFragment** object has the following properties:

**masterDoc**
: This property is expected to be a **Document**

# Object Document

**Document** has the all the properties and methods of **DocumentFragment** as well as the properties and methods defined below.

The **Document** object has the following properties:

**doctype**
: This property is expected to be a **DocumentType**

**implementation**
: This property is expected to be a **DOMImplementation**

**documentElement**
: This property is expected to be a **Element**

**createDocumentFragment**
: This property is expected to be a **DocumentFragment**

**createEntity**
This property is expected to be a **Entity**
**createEntityReference**
This property is expected to be a **EntityReference**

The **Document** object has the following methods:

**createElement(tagName)**
This method returns a **Element**. The **tagName** parameter is expected to be of type **wstring.**.
**createTextNode(data)**
This method returns a **Text**. The **data** parameter is expected to be of type **wstring.**.
**createComment(data)**
This method returns a **Comment**. The **data** parameter is expected to be of type **wstring.**.
**createCDATASection(data)**
This method returns a **CDATASection**. The **data** parameter is expected to be of type **wstring.**.
**createProcessingInstruction(target, data)**
This method returns a **ProcessingInstruction**. The **target** parameter is expected to be of type **wstring.**. The **data** parameter is expected to be of type **wstring.**.
**createAttribute(name)**
This method returns a **Attribute**. The **name** parameter is expected to be of type **wstring.**.
**getElementsByTagName(tagname)**
This method returns a **NodeList**. The **tagname** parameter is expected to be of type **wstring.**.

# Object Node

The **Node** object has the following properties:

**nodeName**
This property is expected to be a **String**
**nodeValue**
This property is expected to be a **String**
**nodeType**
This property is expected to be a **short**
**parentNode**
This property is expected to be a **Node**
**childNodes**
This property is expected to be a **NodeList**
**firstChild**
This property is expected to be a **Node**
**lastChild**
This property is expected to be a **Node**
**previousSibling**
This property is expected to be a **Node**

**nextSibling**
: This property is expected to be a **Node**

**attributes**
: This property is expected to be a **NamedNodeMap**

**hasChildNodes**
: This property is expected to be a **boolean**

The **Node** object has the following methods:

**insertBefore(newChild, refChild)**
: This method returns a **Node**. The **newChild** parameter is expected to be of type **Node.**. The **refChild** parameter is expected to be of type **Node.**.

**replaceChild(newChild, oldChild)**
: This method returns a **Node**. The **newChild** parameter is expected to be of type **Node.**. The **oldChild** parameter is expected to be of type **Node.**.

**removeChild(oldChild)**
: This method returns a **Node**. The **oldChild** parameter is expected to be of type **Node.**.

**appendChild(newChild)**
: This method returns a **Node**. The **newChild** parameter is expected to be of type **Node.**.

**cloneNode(deep)**
: This method returns a **Node**. The **deep** parameter is expected to be of type **boolean.**.

**equals(arg, deep)**
: This method returns a **boolean**. The **arg** parameter is expected to be of type **Node.**. The **deep** parameter is expected to be of type **boolean.**.

# Object NodeList

The **NodeList** object has the following properties:

**size**
: This property is expected to be a **int**

The **NodeList** object has the following methods:

**item(index)**
: This method returns a **Node**. The **index** parameter is expected to be of type **unsigned long.**.

# Object NamedNodeMap

The **NamedNodeMap** object has the following properties:

**size**
: This property is expected to be a **int**

The **NamedNodeMap** object has the following methods:

**getNamedItem(name)**
: This method returns a **Node**. The **name** parameter is expected to be of type **wstring.**.

**setNamedItem(arg)**
: This method returns a **void**. The **arg** parameter is expected to be of type **Node.**.

**removeNamedItem(name)**
: This method returns a **Node**. The **name** parameter is expected to be of type **wstring.**.

**item(index)**
: This method returns a **Node**. The **index** parameter is expected to be of type **unsigned long.**.

# Object Data

**Data** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Data** object has the following properties:

**data**
: This property is expected to be a **String**

**size**
: This property is expected to be a **int**

The **Data** object has the following methods:

**substring(start, count)**
: This method returns a **wstring**. The **start** parameter is expected to be of type **unsigned long.**. The **count** parameter is expected to be of type **unsigned long.**.

**append(arg)**
: This method returns a **void**. The **arg** parameter is expected to be of type **wstring.**.

**insert(offset, arg)**
: This method returns a **void**. The **offset** parameter is expected to be of type **unsigned long.**. The **arg** parameter is expected to be of type **wstring.**.

**delete(offset, count)**
: This method returns a **void**. The **offset** parameter is expected to be of type **unsigned long.**. The **count** parameter is expected to be of type **unsigned long.**.

**replace(offset, count, arg)**
: This method returns a **void**. The **offset** parameter is expected to be of type **unsigned long.**. The **count** parameter is expected to be of type **unsigned long.**. The **arg** parameter is expected to be of type **wstring.**.

# Object Attribute

**Attribute** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Attribute** object has the following properties:

**specified**
: This property is expected to be a **boolean**

**name**
: This property is expected to be a **wstring**

**value**
: This property is expected to be a **wstring**

# Object Element

**Element** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Element** object has the following properties:

**tagName**
: This property is expected to be a **wstring**

**attributes**
: This property is expected to be a **NamedNodeMap**

**normalize**
: This property is expected to be a **void**

The **Element** object has the following methods:

**getAttribute(name)**
: This method returns a **wstring**. The **name** parameter is expected to be of type **wstring.**.

**setAttribute(name, value)**
: This method returns a **void**. The **name** parameter is expected to be of type **string.**. The **value** parameter is expected to be of type **string.**.

**removeAttribute(name)**
: This method returns a **void**. The **name** parameter is expected to be of type **wstring.**.

**getAttributeNode(name)**
: This method returns a **Attribute**. The **name** parameter is expected to be of type **wstring.**.

**setAttributeNode(newAttr)**
: This method returns a **void**. The **newAttr** parameter is expected to be of type **Attribute.**.

**removeAttributeNode(oldAttr)**
: This method returns a **void**. The **oldAttr** parameter is expected to be of type **Attribute.**.

**getElementsByTagName(tagname)**
: This method returns a **NodeList**. The **tagname** parameter is expected to be of type **wstring.**.

# Object Text

**Text** has the all the properties and methods of **Data** as well as the properties and methods defined below.

The **Text** object has the following methods:

**splitText(offset)**
: This method returns a **Text**. The **offset** parameter is expected to be of type **unsigned long.**.

**joinText(node1, node2)**
: This method returns a **Text**. The **node1** parameter is expected to be of type **Text.**. The **node2** parameter is expected to be of type **Text.**.

# Object Comment

**Comment** has the all the properties and methods of **Data** as well as the properties and methods defined below.

# Object ProcessingInstruction

**ProcessingInstruction** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **ProcessingInstruction** object has the following properties:

**target**
: This property is expected to be a **String**

**data**
: This property is expected to be a **String**

# Object CDATASection

**CDATASection** has the all the properties and methods of **Text** as well as the properties and methods defined below.

# Object DocumentType

**DocumentType** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **DocumentType** object has the following properties:

**name**
: This property is expected to be a **String**

**entities**
This property is expected to be a **NamedNodeMap**
**notations**
This property is expected to be a **NamedNodeMap**

# Object Notation

**Notation** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Notation** object has the following properties:

**publicId**
This property is expected to be a **String**
**systemId**
This property is expected to be a **String**

# Object Entity

**Entity** has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Entity** object has the following properties:

**publicId**
This property is expected to be a **String**
**systemId**
This property is expected to be a **String**
**notationName**
This property is expected to be a **String**

# Object EntityReference

**EntityReference** has the all the properties and methods of **Node** as well as the properties and methods defined below.

# D.2: Document Object Model Level 1 HTML Bindings

## D.2.1: OMG IDL Definitions for the Level 1 HTML DOM

```
interface HTMLCollection {
  readonly attribute  long                length;
  Node                        item(in long index);
  Node                        namedItem(in wstring name);
};

interface HTMLDocument : Document {
```

```
           attribute  wstring             title;
  readonly attribute  wstring             referrer;
  readonly attribute  wstring             fileSize;
  readonly attribute  wstring             fileCreatedDate;
  readonly attribute  wstring             fileModifiedDate;
  readonly attribute  wstring             fileUpdatedDate;
  readonly attribute  wstring             domain;
  readonly attribute  wstring             URL;
           attribute  HTMLElement         body;
  readonly attribute  HTMLCollection      images;
  readonly attribute  HTMLCollection      applets;
  readonly attribute  HTMLCollection      links;
  readonly attribute  HTMLCollection      forms;
  readonly attribute  HTMLCollection      anchors;
           attribute  wstring             cookie;
  void                     open();
  void                     close();
  void                     write(in wstring text);
  void                     writeln(in wstring text);
  Element                  getElementById(in wstring elementId);
  NodeList                 getElementsByName(in wstring elementName);
};

interface HTMLElement : Element {
           attribute  wstring             id;
           attribute  wstring             title;
           attribute  wstring             lang;
           attribute  wstring             dir;
           attribute  wstring             className;
};

interface HTMLHtmlElement : HTMLElement {
           attribute  wstring             version;
};

interface HTMLHeadElement : HTMLElement {
           attribute  wstring             profile;
};

interface HTMLLinkElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  wstring             charset;
           attribute  wstring             href;
           attribute  wstring             hreflang;
           attribute  wstring             media;
           attribute  wstring             rel;
           attribute  wstring             rev;
           attribute  wstring             target;
           attribute  wstring             type;
};

interface HTMLTitleElement : HTMLElement {
           attribute  wstring             text;
};

interface HTMLMetaElement : HTMLElement {
           attribute  wstring             content;
```

```
           attribute  wstring             httpEquiv;
           attribute  wstring             name;
           attribute  wstring             scheme;
};

interface HTMLBaseElement : HTMLElement {
           attribute  wstring             href;
           attribute  wstring             target;
};

interface HTMLIsIndexElement : HTMLElement {
  readonly attribute  HTMLFormElement     form;
           attribute  wstring             prompt;
};

interface HTMLStyleElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  wstring             media;
           attribute  wstring             type;
};

interface HTMLBodyElement : HTMLElement {
           attribute  wstring             aLink;
           attribute  wstring             background;
           attribute  wstring             bgColor;
           attribute  wstring             link;
           attribute  wstring             text;
           attribute  wstring             vLink;
};

interface HTMLFormElement : HTMLElement {
  readonly attribute  HTMLCollection      elements;
  readonly attribute  long                length;
           attribute  wstring             name;
           attribute  wstring             acceptCharset;
           attribute  wstring             action;
           attribute  wstring             enctype;
           attribute  wstring             method;
           attribute  wstring             target;
};

interface HTMLSelectElement : HTMLElement {
  readonly attribute  wstring             type;
           attribute  long                selectedIndex;
           attribute  wstring             value;
           attribute  long                length;
  readonly attribute  HTMLFormElement     form;
           attribute  HTMLCollection      options;
           attribute  boolean             disabled;
           attribute  boolean             multiple;
           attribute  wstring             name;
           attribute  long                size;
           attribute  long                tabIndex;
  void                     add(in HTMLElement element,
                               in HTMLElement before);
  void                     remove(in long index);
  void                     blur();
```

```
  void                    focus();
};

interface HTMLOptGroupElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  wstring             label;
};

interface HTMLOptionElement : HTMLElement {
  readonly attribute  HTMLFormElement     form;
           attribute  boolean             defaultSelected;
  readonly attribute  wstring            text;
           attribute  long                index;
           attribute  boolean             disabled;
           attribute  wstring             label;
  readonly attribute  boolean             selected;
           attribute  wstring             value;
};

interface HTMLInputElement : HTMLElement {
           attribute  wstring             defaultValue;
           attribute  boolean             defaultChecked;
  readonly attribute  HTMLFormElement     form;
           attribute  wstring             accept;
           attribute  wstring             accessKey;
           attribute  wstring             align;
           attribute  wstring             alt;
           attribute  boolean             checked;
           attribute  boolean             disabled;
           attribute  long                maxLength;
           attribute  wstring             name;
           attribute  boolean             readOnly;
           attribute  wstring             size;
           attribute  wstring             src;
           attribute  long                tabIndex;
  readonly attribute  wstring             type;
           attribute  wstring             useMap;
           attribute  wstring             value;
  void                    blur();
  void                    focus();
  void                    select();
  void                    click();
};

interface HTMLTextAreaElement : HTMLElement {
           attribute  wstring             defaultValue;
  readonly attribute  HTMLFormElement     form;
           attribute  wstring             accessKey;
           attribute  long                cols;
           attribute  boolean             disabled;
           attribute  wstring             name;
           attribute  boolean             readOnly;
           attribute  long                rows;
           attribute  long                tabIndex;
  readonly attribute  wstring             type;
  void                    blur();
  void                    focus();
```

```
  void                    select();
};

interface HTMLButtonElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accessKey;
           attribute  boolean              disabled;
           attribute  wstring              name;
           attribute  long                 tabIndex;
  readonly attribute  wstring              type;
           attribute  wstring              value;
};

interface HTMLLabelElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accessKey;
           attribute  wstring              htmlFor;
};

interface HTMLFieldSetElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
};

interface HTMLLegendElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              accessKey;
           attribute  wstring              align;
};

interface HTMLUListElement : HTMLElement {
           attribute  boolean              compact;
           attribute  wstring              type;
};

interface HTMLOListElement : HTMLElement {
           attribute  boolean              compact;
           attribute  long                 start;
           attribute  wstring              type;
};

interface HTMLDListElement : HTMLElement {
           attribute  boolean              compact;
};

interface HTMLDirectoryElement : HTMLElement {
           attribute  boolean              compact;
};

interface HTMLMenuElement : HTMLElement {
           attribute  boolean              compact;
};

interface HTMLLIElement : HTMLElement {
           attribute  wstring              type;
           attribute  long                 value;
};
```

```
interface HTMLBlockquoteElement : HTMLElement {
           attribute  wstring             cite;
};

interface HTMLDivElement : HTMLElement {
           attribute  wstring             align;
};

interface HTMLParagraphElement : HTMLElement {
           attribute  wstring             align;
};

interface HTMLHeadingElement : HTMLElement {
           attribute  wstring             align;
};

interface HTMLQuoteElement : HTMLElement {
           attribute  wstring             cite;
};

interface HTMLPreElement : HTMLElement {
           attribute  long                width;
};

interface HTMLBRElement : HTMLElement {
           attribute  wstring             clear;
};

interface HTMLBaseFontElement : HTMLElement {
           attribute  wstring             color;
           attribute  wstring             face;
           attribute  wstring             size;
};

interface HTMLFontElement : HTMLElement {
           attribute  wstring             color;
           attribute  wstring             face;
           attribute  wstring             size;
};

interface HTMLHRElement : HTMLElement {
           attribute  wstring             align;
           attribute  boolean             noShade;
           attribute  wstring             size;
           attribute  wstring             width;
};

interface HTMLInsElement : HTMLElement {
           attribute  wstring             cite;
           attribute  wstring             dateTime;
};

interface HTMLDelElement : HTMLElement {
           attribute  wstring             cite;
           attribute  wstring             dateTime;
};
```

```
interface HTMLModElement : HTMLElement {
           attribute  wstring              cite;
           attribute  wstring              dateTime;
};

interface HTMLAnchorElement : HTMLElement {
           attribute  wstring              accessKey;
           attribute  wstring              charset;
           attribute  wstring              coords;
           attribute  wstring              href;
           attribute  wstring              hreflang;
           attribute  wstring              name;
           attribute  wstring              rel;
           attribute  wstring              rev;
           attribute  wstring              shape;
           attribute  long                 tabIndex;
           attribute  wstring              target;
           attribute  wstring              type;
  void                     blur();
  void                     focus();
};

interface HTMLImageElement : HTMLElement {
           attribute  wstring              lowSrc;
           attribute  wstring              name;
           attribute  wstring              align;
           attribute  wstring              alt;
           attribute  wstring              border;
           attribute  wstring              height;
           attribute  wstring              hspace;
           attribute  boolean              isMap;
           attribute  wstring              longDesc;
           attribute  wstring              src;
           attribute  wstring              useMap;
           attribute  wstring              vspace;
           attribute  wstring              width;
};

interface HTMLObjectElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  wstring              code;
           attribute  wstring              align;
           attribute  wstring              archive;
           attribute  wstring              border;
           attribute  wstring              codeBase;
           attribute  wstring              codeType;
           attribute  wstring              data;
           attribute  boolean              declare;
           attribute  wstring              height;
           attribute  wstring              hspace;
           attribute  wstring              name;
           attribute  wstring              standby;
           attribute  long                 tabIndex;
           attribute  wstring              type;
           attribute  wstring              useMap;
           attribute  wstring              vspace;
           attribute  wstring              width;
```

```
};

interface HTMLParamElement : HTMLElement {
           attribute  wstring             name;
           attribute  wstring             type;
           attribute  wstring             value;
           attribute  wstring             valueType;
};

interface HTMLAppletElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             alt;
           attribute  wstring             archive;
           attribute  wstring             code;
           attribute  wstring             codeBase;
           attribute  wstring             height;
           attribute  wstring             hspace;
           attribute  wstring             name;
           attribute  wstring             object;
           attribute  wstring             vspace;
           attribute  wstring             width;
};

interface HTMLMapElement : HTMLElement {
  readonly attribute  HTMLCollection      areas;
           attribute  wstring             name;
};

interface HTMLAreaElement : HTMLElement {
           attribute  wstring             accessKey;
           attribute  wstring             alt;
           attribute  wstring             coords;
           attribute  wstring             href;
           attribute  boolean             noHref;
           attribute  wstring             shape;
           attribute  long                tabIndex;
           attribute  wstring             target;
};

interface HTMLScriptElement : HTMLElement {
           attribute  wstring             text;
           attribute  wstring             htmlFor;
           attribute  wstring             event;
           attribute  wstring             charset;
           attribute  boolean             defer;
           attribute  wstring             src;
           attribute  wstring             type;
};

interface HTMLTableElement : HTMLElement {
           attribute  HTMLTableCaptionElement caption;
           attribute  HTMLTableSectionElement tHead;
           attribute  HTMLTableSectionElement tFoot;
  readonly attribute  HTMLCollection      rows;
           attribute  HTMLCollection      tBodies;
           attribute  wstring             align;
           attribute  wstring             bgColor;
```

```
           attribute  wstring             border;
           attribute  wstring             cellPadding;
           attribute  wstring             cellSpacing;
           attribute  wstring             frame;
           attribute  wstring             rules;
           attribute  wstring             summary;
           attribute  wstring             width;
  HTMLElement              createTHead();
  void                     deleteTHead();
  HTMLElement              createTFoot();
  void                     deleteTFoot();
  HTMLElement              createCaption();
  void                     deleteCaption();
  HTMLElement              insertRow(in long index);
  void                     deleteRow(in long index);
};

interface HTMLTableCaptionElement : HTMLElement {
           attribute  wstring             align;
};

interface HTMLTableColElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  long                span;
           attribute  wstring             vAlign;
           attribute  wstring             width;
};

interface HTMLTheadElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
};

interface HTMLTbodyElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
};

interface HTMLTfootElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
};

interface HTMLTableSectionElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             vAlign;
           attribute  HTMLCollection      rows;
  HTMLElement              insertRow(in long index);
  void                     deleteRow(in long index);
```

```
};

interface HTMLTableRowElement : HTMLElement {
           attribute  long                rowIndex;
           attribute  long                sectionRowIndex;
           attribute  HTMLCollection      cells;
           attribute  wstring             align;
           attribute  wstring             bgColor;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  wstring             vAlign;
  HTMLElement              insertCell(in long index);
  void                     deleteCell(in long index);
};

interface HTMLTableCellElement : HTMLElement {
           attribute  long                cellIndex;
           attribute  wstring             abbr;
           attribute  wstring             align;
           attribute  wstring             axis;
           attribute  wstring             bgColor;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           attribute  long                colSpan;
           attribute  wstring             headers;
           attribute  wstring             height;
           attribute  boolean             noWrap;
           attribute  long                rowSpan;
           attribute  wstring             scope;
           attribute  wstring             vAlign;
           attribute  wstring             width;
};

interface HTMLFrameSetElement : HTMLElement {
           attribute  wstring             cols;
           attribute  wstring             rows;
};

interface HTMLFrameElement : HTMLElement {
           attribute  wstring             frameBorder;
           attribute  wstring             longDesc;
           attribute  wstring             marginHeight;
           attribute  wstring             marginWidth;
           attribute  wstring             name;
           attribute  boolean             noResize;
           attribute  wstring             scrolling;
           attribute  wstring             src;
};

interface HTMLIFrameElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             frameBorder;
           attribute  wstring             height;
           attribute  wstring             longDesc;
           attribute  wstring             marginHeight;
           attribute  wstring             marginWidth;
           attribute  wstring             name;
```

```
            attribute  wstring             scrolling;
            attribute  wstring             src;
            attribute  wstring             width;
};
```

# D.2.2: Java Definitions for the Level 1 HTML DOM

```
public interface HTMLCollection {
   public long              getLength();

   public Node              item(long index);
   public Node              namedItem(String name);
}

public interface HTMLDocument extends Document {
   public String            getTitle();
   public void              setTitle(String arg);

   public String            getReferrer();

   public String            getFileSize();

   public String            getFileCreatedDate();

   public String            getFileModifiedDate();

   public String            getFileUpdatedDate();

   public String            getDomain();

   public String            getURL();

   public HTMLElement       getBody();
   public void              setBody(HTMLElement arg);

   public HTMLCollection    getImages();

   public HTMLCollection    getApplets();

   public HTMLCollection    getLinks();

   public HTMLCollection    getForms();

   public HTMLCollection    getAnchors();

   public String            getCookie();
   public void              setCookie(String arg);

   public void              open();
   public void              close();
   public void              write(String text);
   public void              writeln(String text);
   public Element           getElementById(String elementId);
   public NodeList          getElementsByName(String elementName);
}
```

```
public interface HTMLElement extends Element {
   public String             getId();
   public void               setId(String arg);

   public String             getTitle();
   public void               setTitle(String arg);

   public String             getLang();
   public void               setLang(String arg);

   public String             getDir();
   public void               setDir(String arg);

   public String             getClassName();
   public void               setClassName(String arg);

}

public interface HTMLHtmlElement extends HTMLElement {
   public String             getVersion();
   public void               setVersion(String arg);

}

public interface HTMLHeadElement extends HTMLElement {
   public String             getProfile();
   public void               setProfile(String arg);

}

public interface HTMLLinkElement extends HTMLElement {
   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public String             getCharset();
   public void               setCharset(String arg);

   public String             getHref();
   public void               setHref(String arg);

   public String             getHreflang();
   public void               setHreflang(String arg);

   public String             getMedia();
   public void               setMedia(String arg);

   public String             getRel();
   public void               setRel(String arg);

   public String             getRev();
   public void               setRev(String arg);

   public String             getTarget();
   public void               setTarget(String arg);

   public String             getType();
   public void               setType(String arg);
```

```
}

public interface HTMLTitleElement extends HTMLElement {
   public String             getText();
   public void               setText(String arg);

}

public interface HTMLMetaElement extends HTMLElement {
   public String             getContent();
   public void               setContent(String arg);

   public String             getHttpEquiv();
   public void               setHttpEquiv(String arg);

   public String             getName();
   public void               setName(String arg);

   public String             getScheme();
   public void               setScheme(String arg);

}

public interface HTMLBaseElement extends HTMLElement {
   public String             getHref();
   public void               setHref(String arg);

   public String             getTarget();
   public void               setTarget(String arg);

}

public interface HTMLIsIndexElement extends HTMLElement {
   public HTMLFormElement    getForm();

   public String             getPrompt();
   public void               setPrompt(String arg);

}

public interface HTMLStyleElement extends HTMLElement {
   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public String             getMedia();
   public void               setMedia(String arg);

   public String             getType();
   public void               setType(String arg);

}

public interface HTMLBodyElement extends HTMLElement {
   public String             getALink();
   public void               setALink(String arg);
```

```
   public String             getBackground();
   public void               setBackground(String arg);

   public String             getBgColor();
   public void               setBgColor(String arg);

   public String             getLink();
   public void               setLink(String arg);

   public String             getText();
   public void               setText(String arg);

   public String             getVLink();
   public void               setVLink(String arg);

}

public interface HTMLFormElement extends HTMLElement {
   public HTMLCollection     getElements();

   public long               getLength();

   public String             getName();
   public void               setName(String arg);

   public String             getAcceptCharset();
   public void               setAcceptCharset(String arg);

   public String             getAction();
   public void               setAction(String arg);

   public String             getEnctype();
   public void               setEnctype(String arg);

   public String             getMethod();
   public void               setMethod(String arg);

   public String             getTarget();
   public void               setTarget(String arg);

}

public interface HTMLSelectElement extends HTMLElement {
   public String             getType();

   public long               getSelectedIndex();
   public void               setSelectedIndex(long arg);

   public String             getValue();
   public void               setValue(String arg);

   public long               getLength();
   public void               setLength(long arg);

   public HTMLFormElement    getForm();

   public HTMLCollection     getOptions();
```

```
   public void               setOptions(HTMLCollection arg);

   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public boolean            getMultiple();
   public void               setMultiple(boolean arg);

   public String             getName();
   public void               setName(String arg);

   public long               getSize();
   public void               setSize(long arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public void               add(HTMLElement element,
                                 HTMLElement before);
   public void               remove(long index);
   public void               blur();
   public void               focus();
}

public interface HTMLOptGroupElement extends HTMLElement {
   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public String             getLabel();
   public void               setLabel(String arg);

}

public interface HTMLOptionElement extends HTMLElement {
   public HTMLFormElement    getForm();

   public boolean            getDefaultSelected();
   public void               setDefaultSelected(boolean arg);

   public String             getText();

   public long               getIndex();
   public void               setIndex(long arg);

   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public String             getLabel();
   public void               setLabel(String arg);

   public boolean            getSelected();

   public String             getValue();
   public void               setValue(String arg);

}
```

```
public interface HTMLInputElement extends HTMLElement {
   public String             getDefaultValue();
   public void               setDefaultValue(String arg);

   public boolean            getDefaultChecked();
   public void               setDefaultChecked(boolean arg);

   public HTMLFormElement    getForm();

   public String             getAccept();
   public void               setAccept(String arg);

   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public String             getAlign();
   public void               setAlign(String arg);

   public String             getAlt();
   public void               setAlt(String arg);

   public boolean            getChecked();
   public void               setChecked(boolean arg);

   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public long               getMaxLength();
   public void               setMaxLength(long arg);

   public String             getName();
   public void               setName(String arg);

   public boolean            getReadOnly();
   public void               setReadOnly(boolean arg);

   public String             getSize();
   public void               setSize(String arg);

   public String             getSrc();
   public void               setSrc(String arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public String             getType();

   public String             getUseMap();
   public void               setUseMap(String arg);

   public String             getValue();
   public void               setValue(String arg);

   public void               blur();
   public void               focus();
   public void               select();
   public void               click();
```

```
}

public interface HTMLTextAreaElement extends HTMLElement {
   public String             getDefaultValue();
   public void               setDefaultValue(String arg);

   public HTMLFormElement    getForm();

   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public long               getCols();
   public void               setCols(long arg);

   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public String             getName();
   public void               setName(String arg);

   public boolean            getReadOnly();
   public void               setReadOnly(boolean arg);

   public long               getRows();
   public void               setRows(long arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public String             getType();

   public void               blur();
   public void               focus();
   public void               select();
}

public interface HTMLButtonElement extends HTMLElement {
   public HTMLFormElement    getForm();

   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public boolean            getDisabled();
   public void               setDisabled(boolean arg);

   public String             getName();
   public void               setName(String arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public String             getType();

   public String             getValue();
   public void               setValue(String arg);

}
```

```
public interface HTMLLabelElement extends HTMLElement {
   public HTMLFormElement    getForm();

   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public String             getHtmlFor();
   public void               setHtmlFor(String arg);

}

public interface HTMLFieldSetElement extends HTMLElement {
   public HTMLFormElement    getForm();

}

public interface HTMLLegendElement extends HTMLElement {
   public HTMLFormElement    getForm();

   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public String             getAlign();
   public void               setAlign(String arg);

}

public interface HTMLUListElement extends HTMLElement {
   public boolean            getCompact();
   public void               setCompact(boolean arg);

   public String             getType();
   public void               setType(String arg);

}

public interface HTMLOListElement extends HTMLElement {
   public boolean            getCompact();
   public void               setCompact(boolean arg);

   public long               getStart();
   public void               setStart(long arg);

   public String             getType();
   public void               setType(String arg);

}

public interface HTMLDListElement extends HTMLElement {
   public boolean            getCompact();
   public void               setCompact(boolean arg);

}

public interface HTMLDirectoryElement extends HTMLElement {
   public boolean            getCompact();
```

```
   public void               setCompact(boolean arg);

}

public interface HTMLMenuElement extends HTMLElement {
   public boolean            getCompact();
   public void               setCompact(boolean arg);

}

public interface HTMLLIElement extends HTMLElement {
   public String             getType();
   public void               setType(String arg);

   public long               getValue();
   public void               setValue(long arg);

}

public interface HTMLBlockquoteElement extends HTMLElement {
   public String             getCite();
   public void               setCite(String arg);

}

public interface HTMLDivElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

}

public interface HTMLParagraphElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

}

public interface HTMLHeadingElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

}

public interface HTMLQuoteElement extends HTMLElement {
   public String             getCite();
   public void               setCite(String arg);

}

public interface HTMLPreElement extends HTMLElement {
   public long               getWidth();
   public void               setWidth(long arg);

}

public interface HTMLBRElement extends HTMLElement {
   public String             getClear();
```

```
   public void               setClear(String arg);

}

public interface HTMLBaseFontElement extends HTMLElement {
   public String             getColor();
   public void               setColor(String arg);

   public String             getFace();
   public void               setFace(String arg);

   public String             getSize();
   public void               setSize(String arg);

}

public interface HTMLFontElement extends HTMLElement {
   public String             getColor();
   public void               setColor(String arg);

   public String             getFace();
   public void               setFace(String arg);

   public String             getSize();
   public void               setSize(String arg);

}

public interface HTMLHRElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

   public boolean            getNoShade();
   public void               setNoShade(boolean arg);

   public String             getSize();
   public void               setSize(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}

public interface HTMLInsElement extends HTMLElement {
   public String             getCite();
   public void               setCite(String arg);

   public String             getDateTime();
   public void               setDateTime(String arg);

}

public interface HTMLDelElement extends HTMLElement {
   public String             getCite();
   public void               setCite(String arg);

   public String             getDateTime();
```

```
   public void               setDateTime(String arg);

}

public interface HTMLModElement extends HTMLElement {
   public String             getCite();
   public void               setCite(String arg);

   public String             getDateTime();
   public void               setDateTime(String arg);

}

public interface HTMLAnchorElement extends HTMLElement {
   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public String             getCharset();
   public void               setCharset(String arg);

   public String             getCoords();
   public void               setCoords(String arg);

   public String             getHref();
   public void               setHref(String arg);

   public String             getHreflang();
   public void               setHreflang(String arg);

   public String             getName();
   public void               setName(String arg);

   public String             getRel();
   public void               setRel(String arg);

   public String             getRev();
   public void               setRev(String arg);

   public String             getShape();
   public void               setShape(String arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public String             getTarget();
   public void               setTarget(String arg);

   public String             getType();
   public void               setType(String arg);

   public void               blur();
   public void               focus();
}

public interface HTMLImageElement extends HTMLElement {
   public String             getLowSrc();
   public void               setLowSrc(String arg);
```

```
    public String             getName();
    public void               setName(String arg);

    public String             getAlign();
    public void               setAlign(String arg);

    public String             getAlt();
    public void               setAlt(String arg);

    public String             getBorder();
    public void               setBorder(String arg);

    public String             getHeight();
    public void               setHeight(String arg);

    public String             getHspace();
    public void               setHspace(String arg);

    public boolean            getIsMap();
    public void               setIsMap(boolean arg);

    public String             getLongDesc();
    public void               setLongDesc(String arg);

    public String             getSrc();
    public void               setSrc(String arg);

    public String             getUseMap();
    public void               setUseMap(String arg);

    public String             getVspace();
    public void               setVspace(String arg);

    public String             getWidth();
    public void               setWidth(String arg);

}

public interface HTMLObjectElement extends HTMLElement {
    public HTMLFormElement    getForm();

    public String             getCode();
    public void               setCode(String arg);

    public String             getAlign();
    public void               setAlign(String arg);

    public String             getArchive();
    public void               setArchive(String arg);

    public String             getBorder();
    public void               setBorder(String arg);

    public String             getCodeBase();
    public void               setCodeBase(String arg);
```

```
   public String             getCodeType();
   public void               setCodeType(String arg);

   public String             getData();
   public void               setData(String arg);

   public boolean            getDeclare();
   public void               setDeclare(boolean arg);

   public String             getHeight();
   public void               setHeight(String arg);

   public String             getHspace();
   public void               setHspace(String arg);

   public String             getName();
   public void               setName(String arg);

   public String             getStandby();
   public void               setStandby(String arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public String             getType();
   public void               setType(String arg);

   public String             getUseMap();
   public void               setUseMap(String arg);

   public String             getVspace();
   public void               setVspace(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}

public interface HTMLParamElement extends HTMLElement {
   public String             getName();
   public void               setName(String arg);

   public String             getType();
   public void               setType(String arg);

   public String             getValue();
   public void               setValue(String arg);

   public String             getValueType();
   public void               setValueType(String arg);

}

public interface HTMLAppletElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);
```

```
   public String             getAlt();
   public void               setAlt(String arg);

   public String             getArchive();
   public void               setArchive(String arg);

   public String             getCode();
   public void               setCode(String arg);

   public String             getCodeBase();
   public void               setCodeBase(String arg);

   public String             getHeight();
   public void               setHeight(String arg);

   public String             getHspace();
   public void               setHspace(String arg);

   public String             getName();
   public void               setName(String arg);

   public String             getObject();
   public void               setObject(String arg);

   public String             getVspace();
   public void               setVspace(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}

public interface HTMLMapElement extends HTMLElement {
   public HTMLCollection     getAreas();

   public String             getName();
   public void               setName(String arg);

}

public interface HTMLAreaElement extends HTMLElement {
   public String             getAccessKey();
   public void               setAccessKey(String arg);

   public String             getAlt();
   public void               setAlt(String arg);

   public String             getCoords();
   public void               setCoords(String arg);

   public String             getHref();
   public void               setHref(String arg);

   public boolean            getNoHref();
   public void               setNoHref(boolean arg);

   public String             getShape();
```

```
   public void               setShape(String arg);

   public long               getTabIndex();
   public void               setTabIndex(long arg);

   public String             getTarget();
   public void               setTarget(String arg);

}

public interface HTMLScriptElement extends HTMLElement {
   public String             getText();
   public void               setText(String arg);

   public String             getHtmlFor();
   public void               setHtmlFor(String arg);

   public String             getEvent();
   public void               setEvent(String arg);

   public String             getCharset();
   public void               setCharset(String arg);

   public boolean            getDefer();
   public void               setDefer(boolean arg);

   public String             getSrc();
   public void               setSrc(String arg);

   public String             getType();
   public void               setType(String arg);

}

public interface HTMLTableElement extends HTMLElement {
   public HTMLTableCaptionElement getCaption();
   public void               setCaption(HTMLTableCaptionElement arg);

   public HTMLTableSectionElement getTHead();
   public void               setTHead(HTMLTableSectionElement arg);

   public HTMLTableSectionElement getTFoot();
   public void               setTFoot(HTMLTableSectionElement arg);

   public HTMLCollection     getRows();

   public HTMLCollection     getTBodies();
   public void               setTBodies(HTMLCollection arg);

   public String             getAlign();
   public void               setAlign(String arg);

   public String             getBgColor();
   public void               setBgColor(String arg);

   public String             getBorder();
   public void               setBorder(String arg);
```

```
   public String             getCellPadding();
   public void               setCellPadding(String arg);

   public String             getCellSpacing();
   public void               setCellSpacing(String arg);

   public String             getFrame();
   public void               setFrame(String arg);

   public String             getRules();
   public void               setRules(String arg);

   public String             getSummary();
   public void               setSummary(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

   public HTMLElement        createTHead();
   public void               deleteTHead();
   public HTMLElement        createTFoot();
   public void               deleteTFoot();
   public HTMLElement        createCaption();
   public void               deleteCaption();
   public HTMLElement        insertRow(long index);
   public void               deleteRow(long index);
}

public interface HTMLTableCaptionElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

}

public interface HTMLTableColElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

   public String             getCh();
   public void               setCh(String arg);

   public String             getChOff();
   public void               setChOff(String arg);

   public long               getSpan();
   public void               setSpan(long arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}

public interface HTMLTheadElement extends HTMLElement {
```

```
   public String             getAlign();
   public void               setAlign(String arg);

   public String             getCh();
   public void               setCh(String arg);

   public String             getChOff();
   public void               setChOff(String arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

}

public interface HTMLTbodyElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

   public String             getCh();
   public void               setCh(String arg);

   public String             getChOff();
   public void               setChOff(String arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

}

public interface HTMLTfootElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

   public String             getCh();
   public void               setCh(String arg);

   public String             getChOff();
   public void               setChOff(String arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

}

public interface HTMLTableSectionElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

   public HTMLCollection     getRows();
   public void               setRows(HTMLCollection arg);

   public HTMLElement        insertRow(long index);
   public void               deleteRow(long index);
}
```

```
public interface HTMLTableRowElement extends HTMLElement {
   public long                getRowIndex();
   public void                setRowIndex(long arg);

   public long                getSectionRowIndex();
   public void                setSectionRowIndex(long arg);

   public HTMLCollection      getCells();
   public void                setCells(HTMLCollection arg);

   public String              getAlign();
   public void                setAlign(String arg);

   public String              getBgColor();
   public void                setBgColor(String arg);

   public String              getCh();
   public void                setCh(String arg);

   public String              getChOff();
   public void                setChOff(String arg);

   public String              getVAlign();
   public void                setVAlign(String arg);

   public HTMLElement         insertCell(long index);
   public void                deleteCell(long index);
}

public interface HTMLTableCellElement extends HTMLElement {
   public long                getCellIndex();
   public void                setCellIndex(long arg);

   public String              getAbbr();
   public void                setAbbr(String arg);

   public String              getAlign();
   public void                setAlign(String arg);

   public String              getAxis();
   public void                setAxis(String arg);

   public String              getBgColor();
   public void                setBgColor(String arg);

   public String              getCh();
   public void                setCh(String arg);

   public String              getChOff();
   public void                setChOff(String arg);

   public long                getColSpan();
   public void                setColSpan(long arg);

   public String              getHeaders();
   public void                setHeaders(String arg);
```

```
   public String             getHeight();
   public void               setHeight(String arg);

   public boolean            getNoWrap();
   public void               setNoWrap(boolean arg);

   public long               getRowSpan();
   public void               setRowSpan(long arg);

   public String             getScope();
   public void               setScope(String arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}

public interface HTMLFrameSetElement extends HTMLElement {
   public String             getCols();
   public void               setCols(String arg);

   public String             getRows();
   public void               setRows(String arg);

}

public interface HTMLFrameElement extends HTMLElement {
   public String             getFrameBorder();
   public void               setFrameBorder(String arg);

   public String             getLongDesc();
   public void               setLongDesc(String arg);

   public String             getMarginHeight();
   public void               setMarginHeight(String arg);

   public String             getMarginWidth();
   public void               setMarginWidth(String arg);

   public String             getName();
   public void               setName(String arg);

   public boolean            getNoResize();
   public void               setNoResize(boolean arg);

   public String             getScrolling();
   public void               setScrolling(String arg);

   public String             getSrc();
   public void               setSrc(String arg);

}
```

```
public interface HTMLIFrameElement extends HTMLElement {
   public String             getAlign();
   public void               setAlign(String arg);

   public String             getFrameBorder();
   public void               setFrameBorder(String arg);

   public String             getHeight();
   public void               setHeight(String arg);

   public String             getLongDesc();
   public void               setLongDesc(String arg);

   public String             getMarginHeight();
   public void               setMarginHeight(String arg);

   public String             getMarginWidth();
   public void               setMarginWidth(String arg);

   public String             getName();
   public void               setName(String arg);

   public String             getScrolling();
   public void               setScrolling(String arg);

   public String             getSrc();
   public void               setSrc(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}
```

## D.2.3: ECMA Script Definitions for the Level 1 HTML DOM

# Object HTMLCollection

The **HTMLCollection** object has the following properties:

**length**
This property is expected to be a **long**

The **HTMLCollection** object has the following methods:

**item(index)**
This method returns a **Node**. The **index** parameter is expected to be of type **long.**.
**namedItem(name)**
This method returns a **Node**. The **name** parameter is expected to be of type **wstring.**.

# Object HTMLDocument

**HTMLDocument** has the all the properties and methods of **Document** as well as the properties and methods defined below.

The **HTMLDocument** object has the following properties:

**title**
: This property is expected to be a **String**

**referrer**
: This property is expected to be a **String**

**fileSize**
: This property is expected to be a **String**

**fileCreatedDate**
: This property is expected to be a **String**

**fileModifiedDate**
: This property is expected to be a **String**

**fileUpdatedDate**
: This property is expected to be a **String**

**domain**
: This property is expected to be a **String**

**URL**
: This property is expected to be a **String**

**body**
: This property is expected to be a **HTMLElement**

**images**
: This property is expected to be a **HTMLCollection**

**applets**
: This property is expected to be a **HTMLCollection**

**links**
: This property is expected to be a **HTMLCollection**

**forms**
: This property is expected to be a **HTMLCollection**

**anchors**
: This property is expected to be a **HTMLCollection**

**cookie**
: This property is expected to be a **String**

**open**
: This property is expected to be a **void**

**close**
: This property is expected to be a **void**

The **HTMLDocument** object has the following methods:

**write(text)**
This method returns a **void**. The **text** parameter is expected to be of type **wstring.**.
**writeln(text)**
This method returns a **void**. The **text** parameter is expected to be of type **wstring.**.
**getElementById(elementId)**
This method returns a **Element**. The **elementId** parameter is expected to be of type **wstring.**.
**getElementsByName(elementName)**
This method returns a **NodeList**. The **elementName** parameter is expected to be of type **wstring.**.

# Object HTMLElement

**HTMLElement** has the all the properties and methods of **Element** as well as the properties and methods defined below.

The **HTMLElement** object has the following properties:

**id**
This property is expected to be a **String**
**title**
This property is expected to be a **String**
**lang**
This property is expected to be a **String**
**dir**
This property is expected to be a **String**
**className**
This property is expected to be a **String**

# Object HTMLHtmlElement

**HTMLHtmlElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHtmlElement** object has the following properties:

**version**
This property is expected to be a **String**

# Object HTMLHeadElement

**HTMLHeadElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHeadElement** object has the following properties:

**profile**
: This property is expected to be a **String**

# Object HTMLLinkElement

**HTMLLinkElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLinkElement** object has the following properties:

**disabled**
: This property is expected to be a **boolean**

**charset**
: This property is expected to be a **String**

**href**
: This property is expected to be a **String**

**hreflang**
: This property is expected to be a **String**

**media**
: This property is expected to be a **String**

**rel**
: This property is expected to be a **String**

**rev**
: This property is expected to be a **String**

**target**
: This property is expected to be a **String**

**type**
: This property is expected to be a **String**

# Object HTMLTitleElement

**HTMLTitleElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTitleElement** object has the following properties:

**text**
: This property is expected to be a **String**

# Object HTMLMetaElement

**HTMLMetaElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLMetaElement** object has the following properties:

**content**
: This property is expected to be a **String**

**httpEquiv**
: This property is expected to be a **String**

**name**
: This property is expected to be a **String**

**scheme**
: This property is expected to be a **String**

# Object HTMLBaseElement

**HTMLBaseElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBaseElement** object has the following properties:

**href**
: This property is expected to be a **String**

**target**
: This property is expected to be a **String**

# Object HTMLIsIndexElement

**HTMLIsIndexElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLIsIndexElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

**prompt**
: This property is expected to be a **String**

# Object HTMLStyleElement

**HTMLStyleElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLStyleElement** object has the following properties:

**disabled**
: This property is expected to be a **boolean**

**media**
This property is expected to be a **String**
**type**
This property is expected to be a **String**

# Object HTMLBodyElement

**HTMLBodyElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBodyElement** object has the following properties:

**aLink**
This property is expected to be a **String**
**background**
This property is expected to be a **String**
**bgColor**
This property is expected to be a **String**
**link**
This property is expected to be a **String**
**text**
This property is expected to be a **String**
**vLink**
This property is expected to be a **String**

# Object HTMLFormElement

**HTMLFormElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFormElement** object has the following properties:

**elements**
This property is expected to be a **HTMLCollection**
**length**
This property is expected to be a **long**
**name**
This property is expected to be a **String**
**acceptCharset**
This property is expected to be a **String**
**action**
This property is expected to be a **String**
**enctype**
This property is expected to be a **String**

**method**
This property is expected to be a **String**
**target**
This property is expected to be a **String**

# Object HTMLSelectElement

**HTMLSelectElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLSelectElement** object has the following properties:

**type**
This property is expected to be a **String**
**selectedIndex**
This property is expected to be a **long**
**value**
This property is expected to be a **String**
**length**
This property is expected to be a **long**
**form**
This property is expected to be a **HTMLFormElement**
**options**
This property is expected to be a **HTMLCollection**
**disabled**
This property is expected to be a **boolean**
**multiple**
This property is expected to be a **boolean**
**name**
This property is expected to be a **String**
**size**
This property is expected to be a **long**
**tabIndex**
This property is expected to be a **long**
**blur**
This property is expected to be a **void**
**focus**
This property is expected to be a **void**

The **HTMLSelectElement** object has the following methods:

**add(element, before)**
This method returns a **void**. The **element** parameter is expected to be of type **HTMLElement.**. The **before** parameter is expected to be of type **HTMLElement.**.
**remove(index)**
This method returns a **void**. The **index** parameter is expected to be of type **long.**.

# Object HTMLOptGroupElement

**HTMLOptGroupElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLOptGroupElement** object has the following properties:

**disabled**
: This property is expected to be a **boolean**

**label**
: This property is expected to be a **String**

# Object HTMLOptionElement

**HTMLOptionElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLOptionElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

**defaultSelected**
: This property is expected to be a **boolean**

**text**
: This property is expected to be a **String**

**index**
: This property is expected to be a **long**

**disabled**
: This property is expected to be a **boolean**

**label**
: This property is expected to be a **String**

**selected**
: This property is expected to be a **boolean**

**value**
: This property is expected to be a **String**

# Object HTMLInputElement

**HTMLInputElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLInputElement** object has the following properties:

**defaultValue**
: This property is expected to be a **String**

**defaultChecked**
This property is expected to be a **boolean**
**form**
This property is expected to be a **HTMLFormElement**
**accept**
This property is expected to be a **String**
**accessKey**
This property is expected to be a **String**
**align**
This property is expected to be a **String**
**alt**
This property is expected to be a **String**
**checked**
This property is expected to be a **boolean**
**disabled**
This property is expected to be a **boolean**
**maxLength**
This property is expected to be a **long**
**name**
This property is expected to be a **String**
**readOnly**
This property is expected to be a **boolean**
**size**
This property is expected to be a **String**
**src**
This property is expected to be a **String**
**tabIndex**
This property is expected to be a **long**
**type**
This property is expected to be a **String**
**useMap**
This property is expected to be a **String**
**value**
This property is expected to be a **String**
**blur**
This property is expected to be a **void**
**focus**
This property is expected to be a **void**
**select**
This property is expected to be a **void**
**click**
This property is expected to be a **void**

# Object HTMLTextAreaElement

**HTMLTextAreaElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTextAreaElement** object has the following properties:

**defaultValue**
: This property is expected to be a **String**

**form**
: This property is expected to be a **HTMLFormElement**

**accessKey**
: This property is expected to be a **String**

**cols**
: This property is expected to be a **long**

**disabled**
: This property is expected to be a **boolean**

**name**
: This property is expected to be a **String**

**readOnly**
: This property is expected to be a **boolean**

**rows**
: This property is expected to be a **long**

**tabIndex**
: This property is expected to be a **long**

**type**
: This property is expected to be a **String**

**blur**
: This property is expected to be a **void**

**focus**
: This property is expected to be a **void**

**select**
: This property is expected to be a **void**

# Object HTMLButtonElement

**HTMLButtonElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLButtonElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

**accessKey**
: This property is expected to be a **String**

**disabled**
: This property is expected to be a **boolean**

**name**
: This property is expected to be a **String**

**tabIndex**
: This property is expected to be a **long**

**type**
: This property is expected to be a **String**

**value**
: This property is expected to be a **String**

# Object HTMLLabelElement

**HTMLLabelElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLabelElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

**accessKey**
: This property is expected to be a **String**

**htmlFor**
: This property is expected to be a **String**

# Object HTMLFieldSetElement

**HTMLFieldSetElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFieldSetElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

# Object HTMLLegendElement

**HTMLLegendElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLegendElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

**accessKey**
This property is expected to be a **String**
**align**
This property is expected to be a **String**

# Object HTMLUListElement

**HTMLUListElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLUListElement** object has the following properties:

**compact**
This property is expected to be a **boolean**
**type**
This property is expected to be a **String**

# Object HTMLOListElement

**HTMLOListElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLOListElement** object has the following properties:

**compact**
This property is expected to be a **boolean**
**start**
This property is expected to be a **long**
**type**
This property is expected to be a **String**

# Object HTMLDListElement

**HTMLDListElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDListElement** object has the following properties:

**compact**
This property is expected to be a **boolean**

# Object HTMLDirectoryElement

**HTMLDirectoryElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDirectoryElement** object has the following properties:

**compact**
: This property is expected to be a **boolean**

# Object HTMLMenuElement

**HTMLMenuElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLMenuElement** object has the following properties:

**compact**
: This property is expected to be a **boolean**

# Object HTMLLIElement

**HTMLLIElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLIElement** object has the following properties:

**type**
: This property is expected to be a **String**

**value**
: This property is expected to be a **long**

# Object HTMLBlockquoteElement

**HTMLBlockquoteElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBlockquoteElement** object has the following properties:

**cite**
: This property is expected to be a **String**

# Object HTMLDivElement

**HTMLDivElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDivElement** object has the following properties:

**align**
: This property is expected to be a **String**

# Object HTMLParagraphElement

**HTMLParagraphElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLParagraphElement** object has the following properties:

**align**
: This property is expected to be a **String**

# Object HTMLHeadingElement

**HTMLHeadingElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHeadingElement** object has the following properties:

**align**
: This property is expected to be a **String**

# Object HTMLQuoteElement

**HTMLQuoteElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLQuoteElement** object has the following properties:

**cite**
: This property is expected to be a **String**

# Object HTMLPreElement

**HTMLPreElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLPreElement** object has the following properties:

**width**
: This property is expected to be a **long**

# Object HTMLBRElement

**HTMLBRElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBRElement** object has the following properties:

**clear**
: This property is expected to be a **String**

# Object HTMLBaseFontElement

**HTMLBaseFontElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBaseFontElement** object has the following properties:

**color**
: This property is expected to be a **String**

**face**
: This property is expected to be a **String**

**size**
: This property is expected to be a **String**

# Object HTMLFontElement

**HTMLFontElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFontElement** object has the following properties:

**color**
: This property is expected to be a **String**

**face**
: This property is expected to be a **String**

**size**
: This property is expected to be a **String**

# Object HTMLHRElement

**HTMLHRElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHRElement** object has the following properties:

**align**
: This property is expected to be a **String**

**noShade**
: This property is expected to be a **boolean**

**size**
: This property is expected to be a **String**

**width**
: This property is expected to be a **String**

# Object HTMLInsElement

**HTMLInsElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLInsElement** object has the following properties:

**cite**
: This property is expected to be a **String**

**dateTime**
: This property is expected to be a **String**

# Object HTMLDelElement

**HTMLDelElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDelElement** object has the following properties:

**cite**
: This property is expected to be a **String**

**dateTime**
: This property is expected to be a **String**

# Object HTMLModElement

**HTMLModElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLModElement** object has the following properties:

**cite**
: This property is expected to be a **String**

**dateTime**
: This property is expected to be a **String**

# Object HTMLAnchorElement

**HTMLAnchorElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLAnchorElement** object has the following properties:

**accessKey**
: This property is expected to be a **String**

**charset**
: This property is expected to be a **String**

**coords**
: This property is expected to be a **String**

**href**
: This property is expected to be a **String**

**hreflang**
: This property is expected to be a **String**

**name**
: This property is expected to be a **String**

**rel**
: This property is expected to be a **String**

**rev**
: This property is expected to be a **String**

**shape**
: This property is expected to be a **String**

**tabIndex**
: This property is expected to be a **long**

**target**
: This property is expected to be a **String**

**type**
: This property is expected to be a **String**

**blur**
: This property is expected to be a **void**

**focus**
: This property is expected to be a **void**

# Object HTMLImageElement

**HTMLImageElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLImageElement** object has the following properties:

**lowSrc**
: This property is expected to be a **String**

**name**
: This property is expected to be a **String**

**align**
: This property is expected to be a **String**

**alt**
: This property is expected to be a **String**

**border**
: This property is expected to be a **String**

**height**
: This property is expected to be a **String**

**hspace**
: This property is expected to be a **String**

**isMap**
: This property is expected to be a **boolean**

**longDesc**
: This property is expected to be a **String**

**src**
: This property is expected to be a **String**

**useMap**
: This property is expected to be a **String**

**vspace**
: This property is expected to be a **String**

**width**
: This property is expected to be a **String**

# Object HTMLObjectElement

**HTMLObjectElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLObjectElement** object has the following properties:

**form**
: This property is expected to be a **HTMLFormElement**

**code**
: This property is expected to be a **String**

**align**
This property is expected to be a **String**
**archive**
This property is expected to be a **String**
**border**
This property is expected to be a **String**
**codeBase**
This property is expected to be a **String**
**codeType**
This property is expected to be a **String**
**data**
This property is expected to be a **String**
**declare**
This property is expected to be a **boolean**
**height**
This property is expected to be a **String**
**hspace**
This property is expected to be a **String**
**name**
This property is expected to be a **String**
**standby**
This property is expected to be a **String**
**tabIndex**
This property is expected to be a **long**
**type**
This property is expected to be a **String**
**useMap**
This property is expected to be a **String**
**vspace**
This property is expected to be a **String**
**width**
This property is expected to be a **String**

# Object HTMLParamElement

**HTMLParamElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLParamElement** object has the following properties:

**name**
This property is expected to be a **String**
**type**
This property is expected to be a **String**

**value**
This property is expected to be a **String**
**valueType**
This property is expected to be a **String**

# Object HTMLAppletElement

**HTMLAppletElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLAppletElement** object has the following properties:

**align**
This property is expected to be a **String**
**alt**
This property is expected to be a **String**
**archive**
This property is expected to be a **String**
**code**
This property is expected to be a **String**
**codeBase**
This property is expected to be a **String**
**height**
This property is expected to be a **String**
**hspace**
This property is expected to be a **String**
**name**
This property is expected to be a **String**
**object**
This property is expected to be a **String**
**vspace**
This property is expected to be a **String**
**width**
This property is expected to be a **String**

# Object HTMLMapElement

**HTMLMapElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLMapElement** object has the following properties:

**areas**
This property is expected to be a **HTMLCollection**

**name**
This property is expected to be a **String**

# Object HTMLAreaElement

**HTMLAreaElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLAreaElement** object has the following properties:

**accessKey**
This property is expected to be a **String**
**alt**
This property is expected to be a **String**
**coords**
This property is expected to be a **String**
**href**
This property is expected to be a **String**
**noHref**
This property is expected to be a **boolean**
**shape**
This property is expected to be a **String**
**tabIndex**
This property is expected to be a **long**
**target**
This property is expected to be a **String**

# Object HTMLScriptElement

**HTMLScriptElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLScriptElement** object has the following properties:

**text**
This property is expected to be a **String**
**htmlFor**
This property is expected to be a **String**
**event**
This property is expected to be a **String**
**charset**
This property is expected to be a **String**
**defer**
This property is expected to be a **boolean**

**src**
This property is expected to be a **String**
**type**
This property is expected to be a **String**

# Object HTMLTableElement

**HTMLTableElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableElement** object has the following properties:

**caption**
This property is expected to be a **HTMLTableCaptionElement**
**tHead**
This property is expected to be a **HTMLTableSectionElement**
**tFoot**
This property is expected to be a **HTMLTableSectionElement**
**rows**
This property is expected to be a **HTMLCollection**
**tBodies**
This property is expected to be a **HTMLCollection**
**align**
This property is expected to be a **String**
**bgColor**
This property is expected to be a **String**
**border**
This property is expected to be a **String**
**cellPadding**
This property is expected to be a **String**
**cellSpacing**
This property is expected to be a **String**
**frame**
This property is expected to be a **String**
**rules**
This property is expected to be a **String**
**summary**
This property is expected to be a **String**
**width**
This property is expected to be a **String**
**createTHead**
This property is expected to be a **HTMLElement**
**deleteTHead**
This property is expected to be a **void**

**createTFoot**
This property is expected to be a **HTMLElement**
**deleteTFoot**
This property is expected to be a **void**
**createCaption**
This property is expected to be a **HTMLElement**
**deleteCaption**
This property is expected to be a **void**

The **HTMLTableElement** object has the following methods:

**insertRow(index)**
This method returns a **HTMLElement**. The **index** parameter is expected to be of type **long.**.
**deleteRow(index)**
This method returns a **void**. The **index** parameter is expected to be of type **long.**.

# Object HTMLTableCaptionElement

**HTMLTableCaptionElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableCaptionElement** object has the following properties:

**align**
This property is expected to be a **String**

# Object HTMLTableColElement

**HTMLTableColElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableColElement** object has the following properties:

**align**
This property is expected to be a **String**
**ch**
This property is expected to be a **String**
**chOff**
This property is expected to be a **String**
**span**
This property is expected to be a **long**
**vAlign**
This property is expected to be a **String**
**width**
This property is expected to be a **String**

# Object HTMLTheadElement

**HTMLTheadElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTheadElement** object has the following properties:

**align**
: This property is expected to be a **String**

**ch**
: This property is expected to be a **String**

**chOff**
: This property is expected to be a **String**

**vAlign**
: This property is expected to be a **String**

# Object HTMLTbodyElement

**HTMLTbodyElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTbodyElement** object has the following properties:

**align**
: This property is expected to be a **String**

**ch**
: This property is expected to be a **String**

**chOff**
: This property is expected to be a **String**

**vAlign**
: This property is expected to be a **String**

# Object HTMLTfootElement

**HTMLTfootElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTfootElement** object has the following properties:

**align**
: This property is expected to be a **String**

**ch**
: This property is expected to be a **String**

**chOff**
: This property is expected to be a **String**

**vAlign**
This property is expected to be a **String**

# Object HTMLTableSectionElement

**HTMLTableSectionElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableSectionElement** object has the following properties:

**align**
This property is expected to be a **String**
**vAlign**
This property is expected to be a **String**
**rows**
This property is expected to be a **HTMLCollection**

The **HTMLTableSectionElement** object has the following methods:

**insertRow(index)**
This method returns a **HTMLElement**. The **index** parameter is expected to be of type **long.**.
**deleteRow(index)**
This method returns a **void**. The **index** parameter is expected to be of type **long.**.

# Object HTMLTableRowElement

**HTMLTableRowElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableRowElement** object has the following properties:

**rowIndex**
This property is expected to be a **long**
**sectionRowIndex**
This property is expected to be a **long**
**cells**
This property is expected to be a **HTMLCollection**
**align**
This property is expected to be a **String**
**bgColor**
This property is expected to be a **String**
**ch**
This property is expected to be a **String**
**chOff**
This property is expected to be a **String**

**vAlign**
This property is expected to be a **String**

The **HTMLTableRowElement** object has the following methods:

**insertCell(index)**
This method returns a **HTMLElement**. The **index** parameter is expected to be of type **long.**.
**deleteCell(index)**
This method returns a **void**. The **index** parameter is expected to be of type **long.**.

# Object HTMLTableCellElement

**HTMLTableCellElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableCellElement** object has the following properties:

**cellIndex**
This property is expected to be a **long**
**abbr**
This property is expected to be a **String**
**align**
This property is expected to be a **String**
**axis**
This property is expected to be a **String**
**bgColor**
This property is expected to be a **String**
**ch**
This property is expected to be a **String**
**chOff**
This property is expected to be a **String**
**colSpan**
This property is expected to be a **long**
**headers**
This property is expected to be a **String**
**height**
This property is expected to be a **String**
**noWrap**
This property is expected to be a **boolean**
**rowSpan**
This property is expected to be a **long**
**scope**
This property is expected to be a **String**
**vAlign**
This property is expected to be a **String**

**width**
This property is expected to be a **String**

# Object HTMLFrameSetElement

**HTMLFrameSetElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFrameSetElement** object has the following properties:

**cols**
This property is expected to be a **String**
**rows**
This property is expected to be a **String**

# Object HTMLFrameElement

**HTMLFrameElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFrameElement** object has the following properties:

**frameBorder**
This property is expected to be a **String**
**longDesc**
This property is expected to be a **String**
**marginHeight**
This property is expected to be a **String**
**marginWidth**
This property is expected to be a **String**
**name**
This property is expected to be a **String**
**noResize**
This property is expected to be a **boolean**
**scrolling**
This property is expected to be a **String**
**src**
This property is expected to be a **String**

# Object HTMLIFrameElement

**HTMLIFrameElement** has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLIFrameElement** object has the following properties:

**align**
: This property is expected to be a **String**

**frameBorder**
: This property is expected to be a **String**

**height**
: This property is expected to be a **String**

**longDesc**
: This property is expected to be a **String**

**marginHeight**
: This property is expected to be a **String**

**marginWidth**
: This property is expected to be a **String**

**name**
: This property is expected to be a **String**

**scrolling**
: This property is expected to be a **String**

**src**
: This property is expected to be a **String**

**width**
: This property is expected to be a **String**

# Appendix E: Index of Objects Defined by the DOM

## Objects defined in Document Object Model (Core) Level 1


Attribute [p.31]
CDATASection [p.37]
Comment [p.36]
DOMImplementation [p.15]
Data [p.28]
Document [p.17]
DocumentFragment [p.15]
DocumentType [p.37]
Element [p.32]
Entity [p.39]
EntityReference [p.39]
NamedNodeMap [p.27]
Node [p.20]
NodeList [p.26]
Notation [p.38]
ProcessingInstruction [p.37]
Text [p.35]


## Objects defined in Document Object Model (HTML) Level 1


HTMLAnchorElement [p.72]
HTMLAppletElement [p.78]
HTMLAreaElement [p.79]
HTMLBRElement [p.69]
HTMLBaseElement [p.52]
HTMLBaseFontElement [p.69]
HTMLBlockquoteElement [p.67]
HTMLBodyElement [p.53]
HTMLButtonElement [p.63]
HTMLCollection [p.43]
HTMLDListElement [p.66]
HTMLDelElement [p.71]
HTMLDirectoryElement [p.66]
HTMLDivElement [p.68]
HTMLDocument [p.44]
HTMLElement [p.49]
HTMLFieldSetElement [p.64]
HTMLFontElement [p.70]
HTMLFormElement [p.54]
HTMLFrameElement [p.91]
HTMLFrameSetElement [p.91]
HTMLHRElement [p.70]
HTMLHeadElement [p.50]
HTMLHeadingElement [p.68]
HTMLHtmlElement [p.50]
HTMLIFrameElement [p.92]
HTMLImageElement [p.74]
HTMLInputElement [p.58]
HTMLInsElement [p.71]
HTMLIsIndexElement [p.52]
HTMLLIElement [p.67]
HTMLLabelElement [p.64]
HTMLLegendElement [p.65]

HTMLLinkElement [p.50]

HTMLMapElement [p.79]

HTMLMenuElement [p.67]

HTMLMetaElement [p.51]

HTMLModElement [p.72]

HTMLOListElement [p.65]

HTMLObjectElement [p.75]

HTMLOptGroupElement [p.57]

HTMLOptionElement [p.58]

HTMLParagraphElement [p.68]

HTMLParamElement [p.77]

HTMLPreElement [p.69]

HTMLQuoteElement [p.68]

HTMLScriptElement [p.80]

HTMLSelectElement [p.55]

HTMLStyleElement [p.53]

HTMLTableCaptionElement [p.84]

HTMLTableCellElement [p.89]

HTMLTableColElement [p.85]

HTMLTableElement [p.81]

HTMLTableRowElement [p.88]

HTMLTableSectionElement [p.87]

HTMLTbodyElement [p.86]

HTMLTextAreaElement [p.61]

HTMLTfootElement [p.87]

HTMLTheadElement [p.85]

HTMLTitleElement [p.51]

HTMLUListElement [p.65]

# Appendix F: Index of Document Object Model Level 1 Definitions


ATTRIBUTE (21)
Attribute (31)
CDATASection (37)
CDATA_SECTION (21)
COMMENT (21)
Comment (36)
DOCUMENT (21)
DOCUMENT_FRAGMENT (21)
DOCUMENT_TYPE (21)
DOMImplementation (15)
Data (28)
Document (17)
DocumentFragment (15)
DocumentType (37)
ELEMENT (21)
ENTITY (21)
ENTITY_REFERENCE (21)
Element (32)
Entity (39)
EntityReference (39)
HTMLAnchorElement (72)
HTMLAppletElement (78)
HTMLAreaElement (79)
HTMLBRElement (69)
HTMLBaseElement (52)
HTMLBaseFontElement (69)
HTMLBlockquoteElement (67)
HTMLBodyElement (53)
HTMLButtonElement (63)
HTMLCollection (43)
HTMLDListElement (66)
HTMLDelElement (71)
HTMLDirectoryElement (66)
HTMLDivElement (68)
HTMLDocument (44)
HTMLElement (49)
HTMLFieldSetElement (64)
HTMLFontElement (70)
HTMLFormElement (54)
HTMLFrameElement (91)
HTMLFrameSetElement (91)
HTMLHRElement (70)
HTMLHeadElement (50)
HTMLHeadingElement (68)
HTMLHtmlElement (50)
HTMLIFrameElement (92)
HTMLImageElement (74)
HTMLInputElement (58)
HTMLInsElement (71)
HTMLIsIndexElement (52)
HTMLLIElement (67)
HTMLLabelElement (64)
HTMLLegendElement (65)
HTMLLinkElement (50)
HTMLMapElement (79)
HTMLMenuElement (67)
HTMLMetaElement (51)
HTMLModElement (72)
HTMLOListElement (65)
HTMLObjectElement (75)
HTMLOptGroupElement (57)
HTMLOptionElement (58)
HTMLParagraphElement (68)
HTMLParamElement (77)
HTMLPreElement (69)
HTMLQuoteElement (68)
HTMLScriptElement (80)
HTMLSelectElement (55)
HTMLStyleElement (53)

HTMLTableCaptionElement (84)
HTMLTableCellElement (89)
HTMLTableColElement (85)
HTMLTableElement (81)
HTMLTableRowElement (88)
HTMLTableSectionElement (87)
HTMLTbodyElement (86)
HTMLTextAreaElement (61)
HTMLTfootElement (87)
HTMLTheadElement (85)
HTMLTitleElement (51)
HTMLUListElement (65)
NamedNodeMap (27)
Node (20)
NodeList (26)
Notation (38)
PROCESSING_INSTRUCTION (21)
ProcessingInstruction (37)
TEXT (21)
Text (35)
URL (45)
aLink (54)
abbr (90)
accept (59)
acceptCharset (55)
accessKey (59) (62) (63) (64) (65) (73) (80)
action (55)
add (56)
align (59) (65) (68) (68) (68) (71) (74) (76) (78) (82) (85) (85) (86) (86) (87) (87) (89) (90) (93)
alt (60) (74) (78) (80)
anchors (46)
append (29)
appendChild (24)
applets (46)
archive (76) (78)
areas (79)
attributes (23)
axis (90)
background (54)
bgColor (54) (82) (89) (90)
blur (57) (61) (62) (73)
body (45)
border (75) (76) (82)
caption (82)
cellIndex (90)
cellPadding (82)
cellSpacing (82)
cells (89)
ch (85) (86) (86) (87) (89) (90)
chOff (85) (86) (86) (87) (89) (90)
charset (51) (73) (81)
checked (60)
childNodes (22)
cite (67) (69) (71) (72) (72)
className (49)
clear (69)
click (61)
cloneNode (25)
close (46)
code (76) (78)
codeBase (76) (78)
codeType (76)
colSpan (90)
color (70) (70)
cols (62) (91)
compact (65) (66) (66) (66) (67)
content (52)
cookie (46)
coords (73) (80)

createAttribute (19)
createCDATASection (19)
createCaption (84)
createComment (18)
createDocumentFragment (18)
createElement (18)
createEntity (19)
createEntityReference (19)
createProcessingInstruction (19)
createTFoot (83)
createTHead (83)
createTextNode (18)
data (29) (37) (76)
dateTime (71) (72) (72)
declare (76)
defaultChecked (59)
defaultSelected (58)
defaultValue (59) (62)
defer (81)
delete (30)
deleteCaption (84)
deleteCell (89)
deleteRow (84) (88)
deleteTFoot (83)
deleteTHead (83)
dir (49)
disabled (51) (53) (56) (57) (58) (60) (62) (63)
doctype (17)
documentElement (17)
domain (45)
elements (54)
enctype (55)
entities (38)
equals (25)
event (81)
face (70) (70)
fileCreatedDate (45)
fileModifiedDate (45)
fileSize (45)
fileUpdatedDate (45)
firstChild (23)
focus (57) (61) (63) (74)
form (53) (56) (58) (59) (62) (63) (64) (64) (65) (76)
forms (46)
frame (83)
frameBorder (92) (93)
getAttribute (33)
getAttributeNode (34)
getAttributes (33)
getElementById (47)
getElementsByName (47)
getElementsByTagName (20) (35)
getName (31)
getNamedItem (27)
getTagName (32)
getValue (32)
hasChildNodes (25)
hasFeature (15)
headers (90)
height (75) (76) (79) (91) (93)
href (51) (52) (73) (80)
hreflang (51) (73)
hspace (75) (76) (79)
htmlFor (64) (81)
httpEquiv (52)
id (49)
images (46)
implementation (17)
index (58)
insert (30)
insertBefore (23)
insertCell (89)
insertRow (84) (88)
isMap (75)
item (26) (28) (43)

joinText (36)
label (57) (58)
lang (49)
lastChild (23)
length (43) (54) (56)
link (54)
links (46)
longDesc (75) (92) (93)
lowSrc (74)
marginHeight (92) (93)
marginWidth (92) (93)
masterDoc (16)
maxLength (60)
media (51) (53)
method (55)
multiple (56)
name (38) (52) (55) (56) (60) (62) (63) (73) (74) (77) (77) (79) (79) (92) (93)
namedItem (44)
nextSibling (23)
noHref (80)
noResize (92)
noShade (71)
noWrap (91)
nodeName (22)
nodeType (22)
nodeValue (22)
normalize (35)
notationName (39)
notations (38)
object (79)
open (46)
options (56)
parentNode (22)
previousSibling (23)
profile (50)
prompt (53)
publicId (38) (39)
readOnly (60) (62)
referrer (45)
rel (51) (73)
remove (57)
removeAttribute (34)
removeAttributeNode (34)
removeChild (24)
removeNamedItem (27)
replace (30)
replaceChild (24)
rev (51) (73)
rowIndex (88)
rowSpan (91)
rows (62) (82) (87) (91)
rules (83)
scheme (52)
scope (91)
scrolling (92) (93)
sectionRowIndex (88)
select (61) (63)
selected (58)
selectedIndex (56)
setAttribute (33)
setAttributeNode (34)
setNamedItem (27)
shape (73) (80)
size (26) (28) (29) (56) (60) (70) (70) (71)
span (85)
specified (31)
splitText (36)
src (60) (75) (81) (92) (93)
standby (77)
start (66)
substring (29)
summary (83)
systemId (39) (39)
tBodies (82)
tFoot (82)
tHead (82)
tabIndex (56) (60) (62) (63) (73) (77) (80)
target (37) (51) (52) (55) (73) (80)

text (51) (54) (58) (81)

title (45) (49)

type (51) (53) (56) (60) (61) (64) (65) (66) (67) (73) (77) (77) (81)

useMap (60) (75) (77)

vAlign (85) (86) (86) (87) (87) (89) (91)

vLink (54)

value (56) (58) (60) (64) (67) (78)

valueType (78)

version (50)

vspace (75) (77) (79)

width (69) (71) (75) (77) (79) (83) (85) (91) (93)

write (47)

writeln (47)