PR-DOM-Level-1-19980818

Document Object Model (DOM) Level 1 Specification

Version 1.0

W3C Proposed Recommendation *18 August, 1998*

This version
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/DOM.ps
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/DOM.tgz
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/DOM.zip
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/DOM.txt
Latest version
http://www.w3.org/TR/PR-DOM-Level-1
Previous versions
http://www.w3.org/TR/1998/WD-DOM-19980720
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

WG Chair
Lauren Wood, *SoftQuad, Inc.*
Editors
Vidur Apparao, *Netscape*
Steve Byrne, *Sun*
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, *Sun (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 of this document

This document is currently undergoing review by the members of the World Wide Web Consortium. It is a stable document derived from a series of working drafts produced over the last year as deliverables of the W3C DOM Activity. Details of this review have been distributed to the Member's representatives.

The review period will end on 15 September, 1998 at 24:00 GMT. Within 14 days from that time, the document's disposition will be announced: it may become a W3C Recommendation (possibly with minor changes), or it may revert to Working Draft status, or it may be dropped as a W3C work item.

As a Proposed Recommendation, this document does not imply any endorsement by the Consortium's staff or Member organizations.

Comments on this document by non-Members should be sent to www-dom@w3.org. We cannot guarantee a personal response but we will try when it is appropriate.

The authors of this document are the DOM WG members, different chapters may have different editors.

Abstract

This specification defines the Document Object Model Level 1, a platform- and language-neutral interface that allows 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 DOM Level 1 specification is separated into two parts: Core and HTML. The Core DOM Level 1 section 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 section 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 HTML Level 1 section provides additional, higher-level interfaces that are used with the fundamental interfaces defined in the Core Level 1 section to provide a more convenient view of an HTML document. A compliant implementation of the HTML DOM implements all of the fundamental Core interfaces as well as the HTML interfaces.

Table of contents

- Expanded Table of Contents 5
- Copyright Notice 7
- What is the Document Object Model? 9

- Chapter 1: Document Object Model (Core) Level 1 15
- Chapter 2: Document Object Model (HTML) Level 1 49

- Appendix A: Contributors 101
- Appendix B: Glossary 103
- Appendix C: IDL Definitions 109
- Appendix D: Java Language Binding 123
- Appendix E: ECMA Script Language Binding 147
- Index 173
- Production Notes 177

Table of contents

Expanded Table of Contents

- Expanded Table of Contents 5
- Copyright Notice 7
- What is the Document Object Model? 9
  - Introduction 10
  - What the Document Object Model is 10
  - What the Document Object Model is not 12
  - Where the Document Object Model came from 12
  - Entities and the DOM Core 12
  - DOM Interfaces and DOM Implementations 13
  - Limitations of Level One 13

- Chapter 1: Document Object Model (Core) Level 1 15
  - 1.1. Overview of the DOM Core Interfaces 16
    - 1.1.1. The DOM Structure Model 16
    - 1.1.2. Memory Management 16
    - 1.1.3. Naming Conventions 17
    - 1.1.4. Inheritance vs Flattened Views of the API 17
    - 1.1.5. The wstring type 18
    - 1.1.6. Case sensitivity in the DOM 18
  - 1.2. Fundamental Interfaces 19
  - 1.3. Extended Interfaces 44
- Chapter 2: Document Object Model (HTML) Level 1 49
  - 2.1. Introduction 50
  - 2.2. HTML Application of Core DOM 50
    - 2.2.1. Naming Conventions 50
  - 2.3. Miscellaneous Object Definitions 51
  - 2.4. Objects related to HTML documents 52
  - 2.5. HTML Elements 55
    - 2.5.1. Property Attributes 55
    - 2.5.2. Naming Exceptions 56
    - 2.5.3. Exposing Element Type Names (tagName) 56
    - 2.5.4. The HTMLElement interface 56
    - 2.5.5. Object definitions 57

- Appendix A: Contributors 101
- Appendix B: Glossary 103
- Appendix C: IDL Definitions 109
  - C.1. Document Object Model Level 1 Core 109
  - C.2. Document Object Model Level 1 HTML 112
- Appendix D: Java Language Binding 123
  - D.1. Document Object Model Level 1 Core 123

- D.2. Document Object Model Level 1 HTML 127
- Appendix E: ECMA Script Language Binding 147
  - E.1. Document Object Model Level 1 Core 147
  - E.2. Document Object Model Level 1 HTML 151
- Index 173
- Production Notes 177
  - 1. The Document Type Definition 178
  - 2. The production process 178
  - 3. Object Definitions 179

Copyright Notice

Copyright © 1998 World Wide Web Consortium , (Massachusetts Institute of Technology , Institut National de Recherche en Informatique et en Automatique , Keio University). All Rights Reserved.

Documents on the W3C site are provided by the copyright holders under the following license. By obtaining, using and/or copying this document, or the W3C document from which this statement is linked, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

1. A link or URI to the original W3C document.
2. The pre-existing copyright notice of the original author, if it doesn't exist, a notice of the form: "Copyright © World Wide Web Consortium , (Massachusetts Institute of Technology , Institut National de Recherche en Informatique et en Automatique , Keio University). All Rights Reserved."
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. In addition, credit shall be attributed to the copyright holders for any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives is granted pursuant to this license.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

Copyright Notice

What is the Document Object Model?

Editors
Jonathan Robie, Texcel Research

Introduction

The Document Object Model (DOM) is an application programming interface (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 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 XML internal and external subsets have not yet been specified.

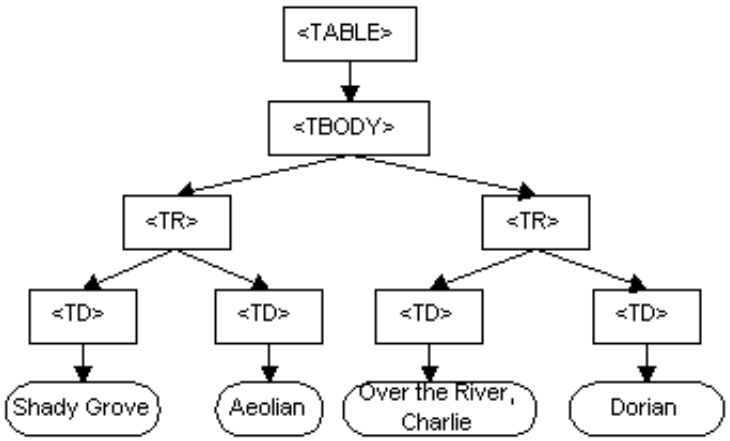
As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used with any programming language. In order to provide a precise, language-independent specification of the DOM 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.. In general, IDLs are designed for specific computing environments. The Document Object Model can be implemented in any computing environment, and does not require the object binding runtimes generally associated with such IDLs.*

What the Document Object Model is

The DOM is a programming API for documents. It closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:

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

The DOM represents this table like this:



DOM representation of the example table

In the DOM, 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 DOM does not specify that documents must be *implemented* as a tree or a grove , nor does it specify how the relationships among objects be implemented. the DOM is a logical model that may be implemented in any convenient manner. 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" 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 DOM 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 that 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. A compliant implementation of the DOM must implement all of the fundamental interfaces in the Core chapter with the semantics as defined. Further, it must implement at least one of the HTML DOM and the

extended (XML) interfaces with the semantics as defined.

What the Document Object Model is not

This section is designed to give a more precise understanding of the DOM 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. DOM programs written in the same language will be source code compatible across platforms, but the DOM 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 DOM 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 W3C Recommendations for these languages. The DOM is a programming model designed to respect these semantics. The DOM 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 DOM.
- 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 DOM 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 DOM 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 DOM Working Group was formed at W3C, 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 DOM 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 an APIs for SGML/XML editors or document repositories, and these object models have also influenced the DOM.

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; 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. 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. The plan is for future Levels of the DOM specification to provide:

1. A structure model for the internal subset and the external subset.
2. Validation against a schema.
3. Control for rendering documents via style sheets.

4. Access control.
5. Thread-safety.
6. Events.

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
Lauren Wood, SoftQuad, Inc.

1.1. Overview of the DOM Core Interfaces

This section defines a minimal set of objects and interfaces for accessing and manipulating document objects. The functionality specified in this section (the *Core* functionality) should be sufficient to allow software developers and web script authors to access and manipulate parsed HTML and XML content inside conforming products. The DOM Core API also allows population of a Document object using only DOM API calls; creating the skeleton Document and saving it persistently is left to the product that implements the DOM API.

1.1.1. The DOM Structure Model

The DOM presents documents as a hierarchy of "Node" objects that also implement other, more specialized interfaces. Some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure. The node types, and which node types they may have as children, are as follows:

- Document -- Element (maximum of one), ProcessingInstruction, Comment, DocumentType
- DocumentFragment -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- DocumentType -- Notation, Entity
- EntityReference -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- Element -- Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- Attribute -- Text, EntityReference
- ProcessingInstruction -- no other nodes
- Comment -- no other nodes
- Text -- no other nodes
- CDATASection -- no other nodes
- Entity -- no other nodes
- Notation -- no other nodes

The DOM also specifies a "NodeList" interface to handle ordered lists of Nodes, such as the children of a Node, or the elements returned by the Element:getElementsByTagName method, and also a NamedNodeMap interface to handle unordered sets of Nodes referenced by their name attribute, such as the Attributes of an Element. NodeLists and NamedNodeMaps in the DOM are "live", that is, changes to the underlying document structure are reflected in all relevant NodeLists and NamedNodeMaps. For example, if a DOM user gets a NodeList object containing the children of an Element, then subsequently adds more children to that element (or removes children, or modifies them), those changes are automatically reflected in the NodeList without further action on the user’s part. Likewise changes to a Node in the tree are reflected in all references to that Node in NodeLists and NamedNodeMaps.

1.1.2. Memory Management

Most of the APIs defined by this specification are *interfaces* rather than classes. That means that an actual implementation need only expose methods with the defined names and specified operation, not actually implement classes that correspond directly to the interfaces. This allows the DOM APIs to be implemented as a thin veneer on top of legacy applications with their own data structures, or on top of newer applications with different class hierarchies. This also means that ordinary constructors (in the Java or C++ sense) cannot be used to create DOM objects, since the underlying objects to be constructed may have little relationship to the DOM interfaces. The conventional solution to this in object-oriented design is to define *factory* methods that create instances of objects that implement the various interfaces. In the DOM Level 1, objects implementing some interface "X" are created by a "createX()" method on the Document interface; this is because all DOM objects live in the context of a specific Document.

The DOM Level 1 API does *not* define a standard way to create DOMImplementation or Document objects; actual DOM implementations must provide some proprietary way of bootstrapping these DOM interfaces, and then all other objects can be built from the Create methods on Document (or by various other convenience methods).

The Core DOM APIs are designed to be compatible with a wide range of languages, including both general-user scripting languages and the more challenging languages used mostly by professional programmers. Thus, the DOM APIs need to operate across a variety of memory management philosophies, from language platforms do not expose memory management to the user at all, through those (notably Java) that provide explicit constructors but provide an automatic garbage collection mechanism to automatically reclaim unused memory, to those (especially C/C++) that generally require the programmer to explicitly allocate object memory, track where it is used, and explicitly free it for re-use. To ensure a consistent API across these platforms, the DOM does not address memory management issues at all, but instead leaves these for the implementation. Neither of the explicit language bindings devised by the DOM Working Group (for ECMAScript and Java) require any memory management methods, but DOM bindings for other languages (especially C or C++) probably will require such support. These extensions will be the responsibility of those adapting the DOM API to a specific language, not the DOM WG.

1.1.3. Naming Conventions

While it would be nice to have attribute and method names that are short, informative, internally consistent, and familiar to users of similar APIs, the names also should not clash with the names in legacy APIs supported by DOM implementations. Furthermore, both OMG IDL and `ECMAScript` have significant limitations in their ability to disambiguate names from different namespaces that makes it difficult to avoid naming conflicts with short, familiar names. So, DOM names tend to be long and quite descriptive in order to be unique across all environments.

The Working Group has also attempted to be internally consistent in its use of various terms, even though these may not be common distinctions in other APIs. For example, we use the method name "remove" when the method changes the structural model, and the method name "delete" when the method gets rid of something inside the structure model. The thing that is deleted is not returned. The thing that is removed may be returned, when it makes sense to return it.

1.1.4. Inheritance vs Flattened Views of the API

The DOM Core APIs present two somewhat different sets of interfaces to an XML/HTML document; one presenting an "object oriented" approach with a hierarchy of inheritance, and a "simplified" view that allows all manipulation to be done via the `Node` interface without requiring casts (in Java and other C-like languages) or query interface calls in COM environments. These operations are fairly expensive in Java and COM, and the DOM may be used in performance-critical environments, so we allow significant functionality using just the Node interface. Because many other users will find the inheritance hierarchy easier to understand than the "everything is a Node" approach to the DOM, we also support the full higher-level interfaces for those who prefer a more object-oriented API.

In practice, this means that there is a certain amount of redundancy in the API. The Working Group considers the "inheritance" approach the primary view of the API, and the full set of functionality on `Node` to be "extra" functionality that users may employ, but that does not eliminate the need for methods on other interfaces that an object-oriented analysis would dictate. (Of course, when the O-O analysis yields an attribute or method that is identical to one on the `Node` interface, we don't specify a completely redundant one). Thus, even though there is a generic `nodeName` attribute on the `Node` interface, there is still a `tagName` attribute on the `Element` interface; these two attributes must contain the same value, but the Working Group considers it worthwhile to support both, given the different constituencies the DOM API must satisfy.

1.1.5. The `wstring` type

To ensure interoperability, the DOM specifies the `wstring` type as follows:

- A `wstring` is a sequence of 16-bit quantities. This may be expressed in IDL terms as:

```
typedef sequence<unsigned short> wstring;
```

- Applications must encode `wstring` using UTF-16 (defined in Appendix C.3 of [UNICODE] and Amendment 1 of [ISO-10646]).The UTF-16 encoding was choosen because of its widespread industry practice. Please note that for both HTML and XML, the document character set (and therefore the notation of numeric character references) is based on UCS-4. A single numeric character reference in a source document may therefore in some cases correspond to two array positions in a `wstring` (a high surrogate and a low surrogate). ***Note:*** *Even though the DOM defines the name of the string type to be `wstring`, bindings may used different names. For, example for Java, `wstring` is bound to the `String` type because it also uses UTF-16 as its encoding.*

Note: *As of August 1998, the OMG IDL specification included a `wstring` type. However, that definition did not meet the interoperability criteria of the DOM API since it relied on encoding negotiation to decide the width of a character.*

1.1.6. Case sensitivity in the DOM

The DOM has many interfaces that imply string matching. HTML processors generally assume an uppercase (less often, lowercase) normalization of names for such things as elements, while XML is explicitly case sensitive. For the purposes of the DOM, string matching takes place on a character code by

character code basis, on the 16 bit value of a `wstring`. As such, the DOM assumes that any normalizations will take place in the processor, *before* the DOM structures are built.

This then raises the issue of exactly what normalizations occur. The W3C I18N working group is in the process of defining exactly which normalizations are necessary for applications implementing 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

INDEX_SIZE_ERR	If index or size is negative, or greater than the allowed value
WSTRING_SIZE_ERR	If the specified range of text does not fit into a wstring
HIERARCHY_REQUEST_ERR	If any node is inserted somewhere it doesn't belong
WRONG_DOCUMENT_ERR	If a node is used in a different document than the one that created it (that doesn't support it)
INVALID_NAME_ERR	If an invalid name is specified
NO_DATA_ALLOWED_ERR	If data is specified for a node which does not support data
NO_MODIFICATION_ALLOWED_ERR	If an attempt is made to modify an object where modifications are not allowed
NOT_FOUND_ERR	If an attempt was made to reference a node in a context where it does not exist
NOT_SUPPORTED_ERR	If the implementation does not support the type of object requested
INUSE_ATTRIBUTE_ERR	If an attempt is made to add an attribute that is already inuse elsewhere

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`.

Implementations may raise other exceptions under other circumstances. For example, implementations may raise an implementation-dependent exception if a `null` argument is passed.

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

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.

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

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. In Level 1, this is the string "1.0" If the version is not specified, supporting any version of the feature will cause the method to return `true`.

Return Values

`true` if the feature is implemented in the specified version, `false` otherwise.
This method raises no exceptions.

Interface *DocumentFragment*

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

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

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

When a `DocumentFragment` is inserted into a `Document` (or indeed any other `Node` that may take children) the children of the `DocumentFragment` and not the `DocumentFragment` itself are inserted into the `Node`. This makes the `DocumentFragment` very useful when the user wishes to create nodes that are siblings; the `DocumentFragment` acts as the parent of these nodes so that the user can use the standard methods from the `Node` interface, such as `insertBefore()` and `appendChild()`.

IDL Definition

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

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.

Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a `Document`, the `Document` interface also contains the factory methods needed to create these objects. The `Node` objects created have a `ownerDocument` attribute which associates them with the `Document` within whose context they were created.

IDL Definition

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

A DOM application may use objects from multiple implementations. This provides access to the `DOMImplementation` object that handles this document.

documentElement

This is a convenience attribute 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 implements the Element interface, so attributes can be specified directly on the returned object.

Parameters

tagName	The name of the element type to instantiate. For XML, this is case-sensitive. For HTML, the `tagName` parameter may be provided in any case, but it must be mapped to the canonical uppercase form by the DOM implementation.

Return Values

A new `Element` object.

Exceptions

DOMException

INVALID_NAME_ERR: Raised if an invalid name is specified.

createDocumentFragment

Create an empty `DocumentFragment` object.

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.

Exceptions

DOMException

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

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.

Exceptions

DOMException

INVALID_NAME_ERR: Raised if an invalid name is specified.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

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.

Exceptions

DOMException

INVALID_NAME_ERR: Raised if an invalid name is specified.

createEntityReference

Creates an EntityReference object.

Parameters

name The name of the entity to reference.

Return Values

The new `EntityReference` object.

Exceptions

DOMException

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`getElementsByTagName`

Returns a `NodeList` of all the `Elements` with a given tag name in the order in which they would be encountered in a preorder traversal of the `Document` tree.

Parameters

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

Return Values

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

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 results in a `DOMException` being raised.

The attributes `nodeName`, `nodeValue` and `attributes` are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (e.g., `nodeValue` for an Element or `attributes` for a Comment), this returns `null`. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

IDL Definition

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

```
  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;
  readonly attribute  Document             ownerDocument;
  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)
                                       raises(DOMException);
  boolean                   hasChildNodes();
  Node                      cloneNode(in boolean deep);
};
```

Definition group *NodeType*

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

ELEMENT_NODE	The node is a `Element`.
ATTRIBUTE_NODE	The node is an `Attribute`.
TEXT_NODE	The node is a `Text` node.
CDATA_SECTION_NODE	The node is a `CDATASection`.
ENTITY_REFERENCE_NODE	The node is an `EntityReference`.
ENTITY_NODE	The node is an `Entity`.
PROCESSING_INSTRUCTION_NODE	The node is a `ProcessingInstruction`.
COMMENT_NODE	The node is a `Comment`.
DOCUMENT_NODE	The node is a `Document`.
DOCUMENT_TYPE_NODE	The node is a `DocumentType`.
DOCUMENT_FRAGMENT_NODE	The node is a `DocumentFragment`.
NOTATION_NODE	The node is a `Notation`.

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

	nodeName	**nodeValue**	**attributes**
Element	tagName	null	NamedNodeMap
Attribute	name of attribute	value of attribute	null
Text	#text	content of the text node	null
CDATASection	#cdata-section	content of the CDATA Section	null
EntityReference	name of entity referenced	null	null
Entity	entity name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentType	document type name	null	null
DocumentFragment	#document-fragment	null	null
Notation	notation name	null	null

Attributes

nodeName

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

nodeValue

The value of a node depends on its type; see the table above. On setting a `NO_MODIFICATION_ALLOWED_ERR DOMException` is raised when the node is readonly. On retrieval a `WSTRING_SIZE_ERR DOMException` is raised when it would return more characters than fit in a `wstring` variable on the implementation platform.

nodeType

A code representing the type of the underlying object's type, as defined above.

parentNode

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

childNodes

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

firstChild

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

lastChild

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

previousSibling

The node immediately preceding the current node. If there is no such node, `null` is returned.

nextSibling

The node immediately following the current node. If there is no such node, this returns `null`.

attributes

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

ownerDocument

Provides access to the `Document` object associated with this `Node`. This is also the Document object used to create new Nodes. When the `Node` is a `Document` this is `null`.

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

Parameters

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

Return Values

The node being inserted.

Exceptions

DOMException

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node.

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

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

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

replaceChild

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

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

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node.

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

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

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

removeChild

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

Parameters

oldChild The node being removed

Return Values

The node removed.

Exceptions

DOMException

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

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

`appendChild`

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

Parameters

newChild The node to add.

If it is a `DocumentFragment` object, the entire contents of the document fragment are moved into the child list of this node

Return Values

The node added.

Exceptions

DOMException

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node.

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

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

`hasChildNodes`

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

Return Values

`true` if the node has any children, `false` if the node has no children.

This method has no parameters.

This method raises no exceptions.

`cloneNode`

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

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

Return Values

The duplicate node.

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.

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        length;
};
```

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

length

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

Interface *NamedNodeMap*

Objects implementing the `NamedNodeMap` interface are used to represent collections of nodes that can be accessed by name. Note that `NamedNodeMap` does not inherit from `NodeList`; `NamedNodeMaps` are not maintained in any particular order. Objects contained in an object implementing `NamedNodeMap` may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a `NamedNodeMap`, and does not imply that the DOM specifies an order to these Nodes. DOM implementations should, when possible, preserve the ordering of objects in a `NamedNodeMap` in case the author of the source document assigned some meaning to this ordering that is not defined in the DOM, XML or HTML specifications.

IDL Definition

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

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.

As the `nodeName` attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a "special" string value) cannot be stored as the names would clash. This is seen as preferable to allowing nodes to be aliased.

Parameters

arg A node to store in a named node list. The node will later be accessible using the value of the `nodeName` attribute of the node. If a node with that name is already present in the list, it is replaced by the new one.

Return Values

If the new Node replaces an existing node with the same name the previously existing Node is returned, otherwise null is returned.

Exceptions

DOMException

WRONG_DOCUMENT_ERR: Raised if `arg` was created from a different document than the one that created the `NamedNodeMap`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this `NamedNodeMap` is readonly.

INUSE_ATTRIBUTE_ERR: Raised if `arg` is an `Attribute` that is already attribute of another `Element` object. The DOM user must explicitly clone `Attribute` nodes to re-use them in other elements.

removeNamedItem

Remove a node identified by its name. If the removed node is an `Attribute` with a default value it is immediately replaced.

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.

Exceptions

DOMException

NOT_FOUND_ERR: Raised if there is no node named `name` in the list.

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

`length`

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

Interface *CharacterData*

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

IDL Definition

```
interface CharacterData : Node {
           attribute  wstring             data;
  readonly attribute  unsigned long       length;
  wstring                  substringData(in unsigned long offset,
                                         in unsigned long count)
                                         raises(DOMException);
  void                     appendData(in wstring arg)
                                      raises(DOMException);
  void                     insertData(in unsigned long offset,
                                      in wstring arg)
                                      raises(DOMException);
  void                     deleteData(in unsigned long offset,
                                      in unsigned long count)
                                      raises(DOMException);
  void                     replaceData(in unsigned long offset,
                                       in unsigned long count,
                                       in wstring arg)
                                       raises(DOMException);
};
```

Attributes

`data`

This provides access to the character data of a node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data data may not fit into a single `wstring`. Attempts to retrieve data that does not fit in a single `wstring` causes a `WSTRING_SIZE_ERR DOMException` to be raised. In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces. In addition, on setting a `NO_MODIFICATION_ALLOWED_ERR DOMException` is raised when the node is readonly.

length

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

Methods

substringData

Extracts a range of data from an object implementing this interface.

Parameters

offset Start offset of substring to extract

count The number of characters to extract.

Return Values

This method returns the specified substring. If the sum of `offset` and `count` exceeds the `length`, then all characters to the end of the data are returned.

Exceptions

DOMException

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

WSTRING_SIZE_ERR: Raised if the specified range of text does not fit into a `wstring`.

appendData

Append the string to the end of the character data in the object implementing this interface. Upon success, `data` provides access to the concatenation of `data` and the `wstring` specified.

Parameters

arg The `wstring` to append.

Exceptions

DOMException

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

insertData

Insert a string at the specified character offset.

Parameters

offset The character offset at which to insert

arg The `wstring` to insert

Exceptions

DOMException

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

deleteData

Remove a range of characters from the node. Upon success, `data` and `length` 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 `length` then all characters from `offset` to the end of the data are deleted.

Exceptions

DOMException

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

"NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

replaceData

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 `length`, 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 must be replaced.

Exceptions

DOMException

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

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

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.

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

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

In XML, the value of an attribute is represented by the child nodes of an `Attribute` node, since the value can contain entity references. Thus, attributes which contain entity references will have a child list containing both `Text` nodes and `EntityReference` nodes. In addition, because the attribute type may be unknown, there are no tokenised attribute values.

IDL Definition

```
interface Attribute : Node {
  readonly attribute  wstring             name;
  readonly attribute  boolean             specified;
           attribute  wstring             value;
};
```

Attributes

name

Returns the name of this attribute.

specified

If this attribute was explicitly given a value in the original document, this is `true`; otherwise, it is `false`. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the specified flag is automatically flipped to `true`. To re-specify the attribute as the default value from the DTD, the user must delete the attribute, and then the implementation will make a new attribute available with specified == `false` and the default value (if one exists).

In summary:

- If the attribute has an assigned value in the document and `specified` is `true`, the value is the assigned value.
- If the attribute has no assigned value in the document and has a default value in the DTD, then `specified` is `false`, and the value is the default value in the DTD.
- If the attribute has no assigned value in the document and has a value of #IMPLIED in the DTD, then the attribute does not appear in the structure model of the document.

value

When used to get the Value of an attribute, returns the value of the attribute as a string. Character and general entity references are replaced with their values in the returned string.

When used to set the Value of an Attribute, creates a Text node with the unparsed contents of the string.

Interface *Element*

By far the vast majority (apart from text) of objects that authors encounter when traversing a document are `Element` nodes. Assume the following XML document:

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

When represented using DOM, the top node is "elementExample", which contains two child `Element` nodes, one for "subelement1" and one for "subelement2". "subelement1" contains no child nodes.

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

IDL Definition

```
interface Element : Node {
  readonly attribute  wstring              tagName;
  wstring                   getAttribute(in wstring name);
  void                      setAttribute(in wstring name,
                                         in wstring value)
                                         raises(DOMException);
  void                      removeAttribute(in wstring name)
                                           raises(DOMException);
  Attribute                 getAttributeNode(in wstring name);
  Attribute                 setAttributeNode(in Attribute newAttr)
                                            raises(DOMException);
  Attribute                 removeAttributeNode(in Attribute oldAttr)
                                               raises(DOMException);
  NodeList                  getElementsByTagName(in wstring name);
  void                      normalize();
};
```

Attributes

tagName

This attribute contains the string that is the element's name. For example, in:

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

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

Methods

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. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string, it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the

implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attribute` node plus any `Text` and `EntityReference` nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.

Parameters

name Name of an attribute

value Value to set in string form

Exceptions

DOMException

INVALID_NAME_ERR: Raised if an invalid name is specified.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

removeAttribute

Removes the `Attribute` with the specified name. If the removed `Attribute` has a default value it is immediately replaced.

Parameters

name The name of the attribute to remove

Exceptions

DOMException

"NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

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 or `null` if there is no such attribute.

This method raises no exceptions.

setAttributeNode

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

Parameters

newAttr	The attribute node to add to the attribute list

Return Values

If the `newAttr` attribute replaces an existing attribute with the same name, the previously existing `Attribute` node is returned, otherwise null is returned.

Exceptions

DOMException

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

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

INUSE_ATTRIBUTE_ERR: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attribute` nodes to re-use them in other elements.

removeAttributeNode

Removes the specified attribute.

Parameters

oldAttr	The `Attribute` node to remove from the attribute list. If the removed `Attribute` has a default value it is immediately replaced.

Return Values

Returns the Attribute node that was removed.

Exceptions

DOMException

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if `oldAttr` is not an attribute of the element.

getElementsByTagName

Returns a `NodeList` of all descendant elements of with a given tag name in the order in which they would be encountered in a preorder traversal of the `Element` tree.

Parameters

name The name of the tag to match on. If the string "*" is given, this method returns all descendant elements of the starting element.

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 full depth of the sub-tree underneath this `Element` into a "normal" form where only markup (e.g., tags, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are no adjacent Text nodes. This can be used to ensure that the DOM view of a document is identical to how it would look if saved and re-loaded, and is useful when operations (such as XPointer lookups) that depend on a particular document tree structure are to be used.

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

Interface *Text*

The `text` interface represents the textual content (termed character data in XML) of an `Element` or `Attribute`. If there is no markup inside an element's content, the text is contained in a single object implementing the `Text` interface that is the child of the element. Any markup is parsed into child elements that are siblings of the text nodes on either side of it, and whose content is 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` merges 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 : CharacterData {
  Text                      splitText(in unsigned long offset)
                                      raises(DOMException);
};
```

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, starting from 0.

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.

Exceptions

DOMException

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

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 : CharacterData {
};
```

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 {
  readonly attribute  wstring             target;
           attribute  wstring             data;
};
```

Attributes

target

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

data

The content of the processing instruction. In HTML this is from from the character immediately after the `<?` to the character immediately preceding the `>`. In XML this is from the first non white space character after the target to the character immediately

preceding the ?>. On setting a NO_MODIFICATION_ALLOWED_ERR DOMException is raised when the node is readonly.

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 to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognised in a CDATA Section is the "]]>" string that ends the CDATA Section. CDATA Sections can not be nested. 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 is contained by the CDATA section. Note that this *may* contain characters that need to be escaped outside of CDATA sections.

The CDATA Section inherits the CharacterData interface through the Text interface. Adjacent CDATA Sections are not merged by use of the Element.normalize() method.

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 clearly understood as of this writing.

IDL Definition

```
interface DocumentType : Node {
  readonly 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 provides access to `foo` and `bar` but not `baz`. All objects supporting the `Node` interface that are accessed through this attribute, also support the `Entity` interface. For HTML, this is always `null`.

notations

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

Interface *Notation*

This interface represents a notation declared in the DTD. A notation either declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification), or is used for formal declaration of Processing Instruction targets (see section 2.6 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 {
  readonly attribute  wstring             publicId;
  readonly 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.

An XML processor may choose to completely expand entities before the structure model is passed to the DOM; in this case there will be no entity references in the document tree.

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.

Level 1 of the DOM API does not support editing `Entity` declarations; if a user wants to make changes to the contents of an Entity, the EntityReference node has to be replaced in the structure model by a clone of the `Entity`'s contents. All the nodes beneath the entity reference are readonly.

IDL Definition

```
interface Entity : Node {
  readonly attribute  wstring             publicId;
  readonly attribute  wstring             systemId;
  readonly 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`.

Interface *EntityReference*

`EntityReference` objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference. Note that character entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. The replacement value of the referenced `Entity`, if available, appears in the child list of the `EntityReference` object. Alternatively, the XML processor may completely expand references to entities while building the structure model, instead of providing `EntityReference` objects.

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.

The resolution of the children of the `EntityReference` (the replacement value of the referenced `Entity`) may be lazily evaluated; actions by the user (such as calling the `childNodes` method on the `EntityReference` Node) are assumed to trigger the evaluation.

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 section extends the Level 1 Core API 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 section; functionality that depends on the specific elements defined in HTML will be found in this section.

The goals of the HTML-specific DOM API are:

- to specialize and add functionality that relates specifically to HTML documents and elements.
- to address issues of backwards compatibility with the "DOM Level 0".
- 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. Interoperability between implementations is only guaranteed for elements and attributes that are specified in these DTDs.

The W3C DOM Working Group attempted, wherever possible, to maintain compatibility with the "DOM Level 0", which refers to 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 DOM Level 1 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 is planned to 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 have corresponding element and attribute definitions 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  unsigned long        length;
  Node                      item(in unsigned 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. The index origin is 0.

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             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).

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.*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.

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. At the time of writing alternate methods for providing similar functionality for both HTML and XML documents were being considered. 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 rendering.

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 "DOM 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 recommend the use of generic methods on the core Element interface for setting, getting and removing attributes.

DTD Data Type	Object Model Data Type
CDATA	wstring
Value list (e.g., (left | right | center))	wstring
one-value Value list (e.g., (border))	boolean
Number	long int

The return value of an attribute that has a 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 is 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 element type names exposed through a property are 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 element interfaces derive from this class. Elements that only expose the HTML 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 the base direction of directionally neutral text and the directionality of tables. 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 style sheet links, and may be used to activate or deactivate style sheets.

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 style sheet object model is planned to 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 style sheet.

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;
  void                      submit();
  void                      reset();
};
```

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.

Methods

submit

Submits the form. It performs the same action as a submit button.

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

reset

Restores a form element's default values. It performs the same action as a reset button.

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

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 designates a source document or message. 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 *HTMLModElement*

Notice of modification to part of a document. See the INS and DEL element definitions in HTML 4.0.

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 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 *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. 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. 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. 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*

An embedded Java applet. 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 only one of each of the `CAPTION`, `THEAD`, and `TFOOT` elements may exist in a table. Therefore, if 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 *HTMLTableSectionElement*

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

IDL Definition

```
interface HTMLTableSectionElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
           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.

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 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: Contributors

Members of the DOM Working Group and Interest Group 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 A: Contributors

Appendix B: Glossary

Editors
Robert S. Sutor, IBM Research

Several of the following term definitions have been borrowed or modified from similar definitions in other W3C or standards documents. See the links within the definitions for more information.

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.

equivalence
Two nodes are *equivalent* if they have the same node type and same node name. Also, if the nodes contain data, that must be the same. Finally, if the nodes have attributes then collection of attribute names must be the same and the attributes corresponding by name must be equivalent as nodes. Two nodes are *deeply equivalent* if they are *equivalent*, the child node lists are equivalent are equivalent as NodeList objects, and the pairs of equivalent attributes must in fact be deeply equivalent. Two NodeList objects are *equivalent* if they have the same length, and the nodes corresponding by index are deeply equivalent. Two NamedNodeMap objects are *equivalent* if they are have the same length, they have same collection of names, and the nodes corresponding by name in the maps are deeply equivalent. Two DocumentType nodes are *equivalent* if they are equivalent as nodes, have the same names, and have equivalent entities and attributes NamedNodeMap objects.

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 B: Glossary

Appendix C: IDL Definitions

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

The IDL files are also available as: http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/idl.zip

C.1: Document Object Model 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.

```
enum ExceptionCode {
  INDEX_SIZE_ERR,
  WSTRING_SIZE_ERR,
  HIERARCHY_REQUEST_ERR,
  WRONG_DOCUMENT_ERR,
  INVALID_NAME_ERR,
  NO_DATA_ALLOWED_ERR,
  NO_MODIFICATION_ALLOWED_ERR,
  NOT_FOUND_ERR,
  NOT_SUPPORTED_ERR,
  INUSE_ATTRIBUTE_ERR
};

exception DOMException {
   ExceptionCode   code;
};

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

interface DocumentFragment : Node {
};

interface Document : Node {
  readonly attribute  DocumentType         doctype;
  readonly attribute  DOMImplementation    implementation;
  readonly attribute  Element              documentElement;
  Element                   createElement(in wstring tagName)
                                         raises(DOMException);
  DocumentFragment          createDocumentFragment();
  Text                      createTextNode(in wstring data);
  Comment                   createComment(in wstring data);
  CDATASection              createCDATASection(in wstring data)
                                           raises(DOMException);
  ProcessingInstruction     createProcessingInstruction(in wstring target,
                                                        in wstring data)
                                                        raises(DOMException);
  Attribute                 createAttribute(in wstring name)
                                          raises(DOMException);
```

```
  EntityReference           createEntityReference(in wstring name)
                                                  raises(DOMException);
  NodeList                  getElementsByTagName(in wstring tagname);
};

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

  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;
  readonly attribute  Document            ownerDocument;
  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)
                                       raises(DOMException);
  boolean                   hasChildNodes();
  Node                      cloneNode(in boolean deep);
};

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

interface NamedNodeMap {
  Node                      getNamedItem(in wstring name);
  Node                      setNamedItem(in Node arg)
                                         raises(DOMException);
  Node                      removeNamedItem(in wstring name)
                                           raises(DOMException);
  Node                      item(in unsigned long index);
```

```
  readonly attribute  unsigned long        length;
};

interface CharacterData : Node {
           attribute  wstring              data;
  readonly attribute  unsigned long        length;
  wstring                   substringData(in unsigned long offset,
                                          in unsigned long count)
                                          raises(DOMException);
  void                      appendData(in wstring arg)
                                       raises(DOMException);
  void                      insertData(in unsigned long offset,
                                       in wstring arg)
                                       raises(DOMException);
  void                      deleteData(in unsigned long offset,
                                       in unsigned long count)
                                       raises(DOMException);
  void                      replaceData(in unsigned long offset,
                                        in unsigned long count,
                                        in wstring arg)
                                        raises(DOMException);
};

interface Attribute : Node {
  readonly attribute  wstring              name;
  readonly attribute  boolean              specified;
           attribute  wstring              value;
};

interface Element : Node {
  readonly attribute  wstring              tagName;
  wstring                   getAttribute(in wstring name);
  void                      setAttribute(in wstring name,
                                         in wstring value)
                                         raises(DOMException);
  void                      removeAttribute(in wstring name)
                                           raises(DOMException);
  Attribute                 getAttributeNode(in wstring name);
  Attribute                 setAttributeNode(in Attribute newAttr)
                                            raises(DOMException);
  Attribute                 removeAttributeNode(in Attribute oldAttr)
                                               raises(DOMException);
  NodeList                  getElementsByTagName(in wstring name);
  void                      normalize();
};

interface Text : CharacterData {
  Text                      splitText(in unsigned long offset)
                                     raises(DOMException);
};

interface Comment : CharacterData {
};

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

```
};

interface CDATASection : Text {
};

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

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

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

interface EntityReference : Node {
};
```

C.2: Document Object Model Level 1 HTML

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

interface HTMLDocument : Document {
           attribute  wstring             title;
  readonly attribute  wstring             referrer;
  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;
  void                     submit();
  void                     reset();
};

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 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 HTMLTableSectionElement : HTMLElement {
           attribute  wstring             align;
           attribute  wstring             ch;
           attribute  wstring             chOff;
```

```
           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;
};
```

C.2: Document Object Model Level 1 HTML

Appendix D: Java Language Binding

This appendix contains the complete Java binding for the Level 1 Document Object Model. The definitions are divided into Core and HTML.

The Java files are also available as
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/java-binding.zip

D.1: Document Object Model Level 1 Core

```
 public static final int INDEX_SIZE_ERR= 1;
 public static final int WSTRING_SIZE_ERR= 2;
 public static final int HIERARCHY_REQUEST_ERR= 3;
 public static final int WRONG_DOCUMENT_ERR= 4;
 public static final int INVALID_NAME_ERR= 5;
 public static final int NO_DATA_ALLOWED_ERR= 6;
 public static final int NO_MODIFICATION_ALLOWED_ERR= 7;
 public static final int NOT_FOUND_ERR= 8;
 public static final int NOT_SUPPORTED_ERR= 9;
 public static final int INUSE_ATTRIBUTE_ERR= 10;


public abstract class DOMException extends RuntimeException {
   public DOMException(int code, String message) {
      super(message);
      this.code = code;
   }

 public static final int INDEX_SIZE_ERR= 1;
 public static final int WSTRING_SIZE_ERR= 2;
 public static final int HIERARCHY_REQUEST_ERR= 3;
 public static final int WRONG_DOCUMENT_ERR= 4;
 public static final int INVALID_NAME_ERR= 5;
 public static final int NO_DATA_ALLOWED_ERR= 6;
 public static final int NO_MODIFICATION_ALLOWED_ERR= 7;
 public static final int NOT_FOUND_ERR= 8;
 public static final int NOT_SUPPORTED_ERR= 9;
 public static final int INUSE_ATTRIBUTE_ERR= 10;


   public int   code;
}

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

public interface DocumentFragment extends Node {
}

public interface Document extends Node {
   public DocumentType       getDoctype();
```

```
   public DOMImplementation  getImplementation();

   public Element            getDocumentElement();

   public Element            createElement(String tagName)
                                        throws DOMException;
   public DocumentFragment   createDocumentFragment();
   public Text               createTextNode(String data);
   public Comment            createComment(String data);
   public CDATASection       createCDATASection(String data)
                                           throws DOMException;
   public ProcessingInstruction createProcessingInstruction(String target,
                                                             String data)
                                                             throws DOMException;
   public Attribute          createAttribute(String name)
                                         throws DOMException;
   public EntityReference    createEntityReference(String name)
                                            throws DOMException;
   public NodeList           getElementsByTagName(String tagname);
}

public interface Node {
   // NodeType
   public static final short           ELEMENT_NODE         = 1;
   public static final short           ATTRIBUTE_NODE       = 2;
   public static final short           TEXT_NODE            = 3;
   public static final short           CDATA_SECTION_NODE   = 4;
   public static final short           ENTITY_REFERENCE_NODE = 5;
   public static final short           ENTITY_NODE          = 6;
   public static final short           PROCESSING_INSTRUCTION_NODE = 7;
   public static final short           COMMENT_NODE         = 8;
   public static final short           DOCUMENT_NODE        = 9;
   public static final short           DOCUMENT_TYPE_NODE   = 10;
   public static final short           DOCUMENT_FRAGMENT_NODE = 11;
   public static final short           NOTATION_NODE        = 12;

   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 Document           getOwnerDocument();

   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)
                                         throws DOMException;
   public boolean            hasChildNodes();
   public Node               cloneNode(boolean deep);
}

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

}

public interface NamedNodeMap {
   public Node               getNamedItem(String name);
   public Node               setNamedItem(Node arg)
                                          throws DOMException;
   public Node               removeNamedItem(String name)
                                            throws DOMException;
   public Node               item(int index);
   public int                getLength();

}

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

   public int                getLength();

   public String             substringData(int offset,
                                           int count)
                                           throws DOMException;
   public void               appendData(String arg)
                                        throws DOMException;
   public void               insertData(int offset,
                                        String arg)
                                        throws DOMException;
   public void               deleteData(int offset,
                                        int count)
                                        throws DOMException;
   public void               replaceData(int offset,
                                         int count,
                                         String arg)
                                         throws DOMException;
}

public interface Attribute extends Node {
```

```
   public String             getName();

   public boolean            getSpecified();

   public String             getValue();
   public void               setValue(String arg);

}

public interface Element extends Node {
   public String             getTagName();

   public String             getAttribute(String name);
   public void               setAttribute(String name,
                                          String value)
                                          throws DOMException;
   public void               removeAttribute(String name)
                                             throws DOMException;
   public Attribute          getAttributeNode(String name);
   public Attribute          setAttributeNode(Attribute newAttr)
                                              throws DOMException;
   public Attribute          removeAttributeNode(Attribute oldAttr)
                                                 throws DOMException;
   public NodeList           getElementsByTagName(String name);
   public void               normalize();
}

public interface Text extends CharacterData {
   public Text               splitText(int offset)
                                       throws DOMException;
}

public interface Comment extends CharacterData {
}

public interface ProcessingInstruction extends Node {
   public String             getTarget();

   public String             getData();
   public void               setData(String arg);

}

public interface CDATASection extends Text {
}

public interface DocumentType extends Node {
   public String             getName();

   public NamedNodeMap       getEntities();

   public NamedNodeMap       getNotations();

}

public interface Notation extends Node {
   public String             getPublicId();
```

```
   public String             getSystemId();

}

public interface Entity extends Node {
   public String             getPublicId();

   public String             getSystemId();

   public String             getNotationName();

}

public interface EntityReference extends Node {
}
```

D.2: Document Object Model Level 1 HTML

```
public interface HTMLCollection {
   public int                getLength();

   public Node               item(int index);
   public Node               namedItem(String name);
}

public interface HTMLDocument extends Document {
   public String             getTitle();
   public void               setTitle(String arg);

   public String             getReferrer();

   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 int                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 void               submit();
   public void               reset();
}

public interface HTMLSelectElement extends HTMLElement {
   public String             getType();

   public int                getSelectedIndex();
   public void               setSelectedIndex(int arg);

   public String             getValue();
   public void               setValue(String arg);
```

```
   public int                getLength();
   public void               setLength(int 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 int                getSize();
   public void               setSize(int arg);

   public int                getTabIndex();
   public void               setTabIndex(int arg);

   public void               add(HTMLElement element,
                                 HTMLElement before);
   public void               remove(int 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 int                getIndex();
   public void               setIndex(int 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 int                getMaxLength();
   public void               setMaxLength(int 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 int                getTabIndex();
   public void               setTabIndex(int 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 int                getCols();
   public void               setCols(int 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 int                getRows();
   public void               setRows(int arg);

   public int                getTabIndex();
   public void               setTabIndex(int 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 int                getTabIndex();
   public void               setTabIndex(int 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 int                getStart();
   public void               setStart(int 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 int                getValue();
   public void               setValue(int 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 int                getWidth();
```

```
   public void               setWidth(int 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 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 int                getTabIndex();
   public void               setTabIndex(int 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 int                getTabIndex();
   public void               setTabIndex(int 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 int                getTabIndex();
   public void               setTabIndex(int 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(int index);
   public void               deleteRow(int 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 int                getSpan();
   public void               setSpan(int arg);

   public String             getVAlign();
   public void               setVAlign(String arg);

   public String             getWidth();
   public void               setWidth(String arg);

}

public interface HTMLTableSectionElement 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 HTMLCollection     getRows();
   public void               setRows(HTMLCollection arg);

   public HTMLElement        insertRow(int index);
   public void               deleteRow(int index);
}

public interface HTMLTableRowElement extends HTMLElement {
   public int                getRowIndex();
   public void               setRowIndex(int arg);

   public int                getSectionRowIndex();
   public void               setSectionRowIndex(int 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(int index);
   public void               deleteCell(int index);
}

public interface HTMLTableCellElement extends HTMLElement {
   public int                getCellIndex();
   public void               setCellIndex(int 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 int                getColSpan();
   public void               setColSpan(int 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 int                getRowSpan();
   public void               setRowSpan(int 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);

}
```

Appendix E: ECMA Script Language Binding

This appendix contains the complete ECMA Script binding for the Level 1 Document Object Model definitions. The definitions are divided into Core and HTML.

E.1: Document Object Model 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 of type **wstring**. The **version** parameter is of type **wstring**.

Object **DocumentFragment**

DocumentFragment has the all the properties and methods of **Node** as well as the properties and methods defined below.

Object **Document**

Document has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Document** object has the following properties:

doctype

This property is of type **DocumentType**.

implementation

This property is of type **DOMImplementation**.

documentElement

This property is of type **Element**.

createDocumentFragment

This property is of type **DocumentFragment**.

The **Document** object has the following methods:

createElement(tagName)

This method returns a **Element**. The **tagName** parameter is of type **wstring**.

createTextNode(data)

This method returns a **Text**. The **data** parameter is of type **wstring**.

createComment(data)

This method returns a **Comment**. The **data** parameter is of type **wstring**.

createCDATASection(data)

This method returns a **CDATASection**. The **data** parameter is of type **wstring**.

createProcessingInstruction(target, data)

This method returns a **ProcessingInstruction**. The **target** parameter is of type **wstring**. The **data** parameter is of type **wstring**.

createAttribute(name)

This method returns a **Attribute**. The **name** parameter is of type **wstring**.

createEntityReference(name)

This method returns a **EntityReference**. The **name** parameter is of type **wstring**.

getElementsByTagName(tagname)
This method returns a **NodeList**. The **tagname** parameter is of type **wstring**.

Object **Node**

The **Node** object has the following properties:

nodeName
This property is of type **String**.

nodeValue
This property is of type **String**.

nodeType
This property is of type **short**.

parentNode
This property is of type **Node**.

childNodes
This property is of type **NodeList**.

firstChild
This property is of type **Node**.

lastChild
This property is of type **Node**.

previousSibling
This property is of type **Node**.

nextSibling
This property is of type **Node**.

attributes
This property is of type **NamedNodeMap**.

ownerDocument
This property is of type **Document**.

hasChildNodes
This property is of type **boolean**.

The **Node** object has the following methods:

insertBefore(newChild, refChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**. The **refChild** parameter is of type **Node**.

replaceChild(newChild, oldChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**. The **oldChild** parameter is of type **Node**.

removeChild(oldChild)
This method returns a **Node**. The **oldChild** parameter is of type **Node**.

appendChild(newChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**.

cloneNode(deep)
This method returns a **Node**. The **deep** parameter is of type **boolean**.

Object **NodeList**

The **NodeList** object has the following properties:

length
This property is of type **int**.

The **NodeList** object has the following methods:

item(index)

This method returns a **Node**. The **index** parameter is of type **unsigned long**.

Object **NamedNodeMap**

The **NamedNodeMap** object has the following properties:

length

This property is of type **int**.

The **NamedNodeMap** object has the following methods:

getNamedItem(name)

This method returns a **Node**. The **name** parameter is of type **wstring**.

setNamedItem(arg)

This method returns a **Node**. The **arg** parameter is of type **Node**.

removeNamedItem(name)

This method returns a **Node**. The **name** parameter is of type **wstring**.

item(index)

This method returns a **Node**. The **index** parameter is of type **unsigned long**.

Object **CharacterData**

CharacterData has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **CharacterData** object has the following properties:

data

This property is of type **String**.

length

This property is of type **int**.

The **CharacterData** object has the following methods:

substringData(offset, count)

This method returns a **wstring**. The **offset** parameter is of type **unsigned long**. The **count** parameter is of type **unsigned long**.

appendData(arg)

This method returns a **void**. The **arg** parameter is of type **wstring**.

insertData(offset, arg)

This method returns a **void**. The **offset** parameter is of type **unsigned long**. The **arg** parameter is of type **wstring**.

deleteData(offset, count)

This method returns a **void**. The **offset** parameter is of type **unsigned long**. The **count** parameter is of type **unsigned long**.

replaceData(offset, count, arg)

This method returns a **void**. The **offset** parameter is of type **unsigned long**. The **count** parameter is of type **unsigned long**. The **arg** parameter is 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:

name

This property is of type **String**.

specified
This property is of type **boolean**.
value
This property is of type **String**.

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 of type **String**.
normalize
This property is of type **void**.

The **Element** object has the following methods:

getAttribute(name)
This method returns a **wstring**. The **name** parameter is of type **wstring**.
setAttribute(name, value)
This method returns a **void**. The **name** parameter is of type **wstring**. The **value** parameter is of type **wstring**.
removeAttribute(name)
This method returns a **void**. The **name** parameter is of type **wstring**.
getAttributeNode(name)
This method returns a **Attribute**. The **name** parameter is of type **wstring**.
setAttributeNode(newAttr)
This method returns a **Attribute**. The **newAttr** parameter is of type **Attribute**.
removeAttributeNode(oldAttr)
This method returns a **Attribute**. The **oldAttr** parameter is of type **Attribute**.
getElementsByTagName(name)
This method returns a **NodeList**. The **name** parameter is of type **wstring**.

Object **Text**

Text has the all the properties and methods of **CharacterData** 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 of type **unsigned long**.

Object **Comment**

Comment has the all the properties and methods of **CharacterData** 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 of type **String**.
data
This property is of type **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 of type **String**.

entities

This property is of type **NamedNodeMap**.

notations

This property is of type **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 of type **String**.

systemId

This property is of type **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 of type **String**.

systemId

This property is of type **String**.

notationName

This property is of type **String**.

Object **EntityReference**

EntityReference has the all the properties and methods of **Node** as well as the properties and methods defined below.

E.2: Document Object Model Level 1 HTML

Object **HTMLCollection**

The **HTMLCollection** object has the following properties:

length

This property is of type **int**.

The **HTMLCollection** object has the following methods:

item(index)

This method returns a **Node**. The **index** parameter is of type **unsigned long**.

namedItem(name)
This method returns a **Node**. The **name** parameter is 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 of type **String**.

referrer
This property is of type **String**.

domain
This property is of type **String**.

URL
This property is of type **String**.

body
This property is of type **HTMLElement**.

images
This property is of type **HTMLCollection**.

applets
This property is of type **HTMLCollection**.

links
This property is of type **HTMLCollection**.

forms
This property is of type **HTMLCollection**.

anchors
This property is of type **HTMLCollection**.

cookie
This property is of type **String**.

open
This property is of type **void**.

close
This property is of type **void**.

The **HTMLDocument** object has the following methods:

write(text)
This method returns a **void**. The **text** parameter is of type **wstring**.

writeln(text)
This method returns a **void**. The **text** parameter is of type **wstring**.

getElementById(elementId)
This method returns a **Element**. The **elementId** parameter is of type **wstring**.

getElementsByName(elementName)
This method returns a **NodeList**. The **elementName** parameter is 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 of type **String**.
title
This property is of type **String**.
lang
This property is of type **String**.
dir
This property is of type **String**.
className
This property is of type **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 of type **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 of type **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 of type **boolean**.
charset
This property is of type **String**.
href
This property is of type **String**.
hreflang
This property is of type **String**.
media
This property is of type **String**.
rel
This property is of type **String**.
rev
This property is of type **String**.
target
This property is of type **String**.
type
This property is of type **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 of type **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 of type **String**.

httpEquiv

This property is of type **String**.

name

This property is of type **String**.

scheme

This property is of type **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 of type **String**.

target

This property is of type **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 of type **HTMLFormElement**.

prompt

This property is of type **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 of type **boolean**.

media

This property is of type **String**.

type

This property is of type **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 of type **String**.

background
This property is of type **String**.

bgColor
This property is of type **String**.

link
This property is of type **String**.

text
This property is of type **String**.

vLink
This property is of type **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 of type **HTMLCollection**.

length
This property is of type **long**.

name
This property is of type **String**.

acceptCharset
This property is of type **String**.

action
This property is of type **String**.

enctype
This property is of type **String**.

method
This property is of type **String**.

target
This property is of type **String**.

submit
This property is of type **void**.

reset
This property is of type **void**.

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 of type **String**.

selectedIndex
This property is of type **long**.

value
This property is of type **String**.
length
This property is of type **long**.
form
This property is of type **HTMLFormElement**.
options
This property is of type **HTMLCollection**.
disabled
This property is of type **boolean**.
multiple
This property is of type **boolean**.
name
This property is of type **String**.
size
This property is of type **long**.
tabIndex
This property is of type **long**.
blur
This property is of type **void**.
focus
This property is of type **void**.

The **HTMLSelectElement** object has the following methods:

add(element, before)
This method returns a **void**. The **element** parameter is of type **HTMLElement**. The **before** parameter is of type **HTMLElement**.
remove(index)
This method returns a **void**. The **index** parameter is 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 of type **boolean**.
label
This property is of type **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 of type **HTMLFormElement**.
defaultSelected
This property is of type **boolean**.
text
This property is of type **String**.

index
This property is of type **long**.
disabled
This property is of type **boolean**.
label
This property is of type **String**.
selected
This property is of type **boolean**.
value
This property is of type **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 of type **String**.
defaultChecked
This property is of type **boolean**.
form
This property is of type **HTMLFormElement**.
accept
This property is of type **String**.
accessKey
This property is of type **String**.
align
This property is of type **String**.
alt
This property is of type **String**.
checked
This property is of type **boolean**.
disabled
This property is of type **boolean**.
maxLength
This property is of type **long**.
name
This property is of type **String**.
readOnly
This property is of type **boolean**.
size
This property is of type **String**.
src
This property is of type **String**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.

useMap
This property is of type **String**.
value
This property is of type **String**.
blur
This property is of type **void**.
focus
This property is of type **void**.
select
This property is of type **void**.
click
This property is of type **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 of type **String**.
form
This property is of type **HTMLFormElement**.
accessKey
This property is of type **String**.
cols
This property is of type **long**.
disabled
This property is of type **boolean**.
name
This property is of type **String**.
readOnly
This property is of type **boolean**.
rows
This property is of type **long**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
blur
This property is of type **void**.
focus
This property is of type **void**.
select
This property is of type **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 of type **HTMLFormElement**.

accessKey
: This property is of type **String**.

disabled
: This property is of type **boolean**.

name
: This property is of type **String**.

tabIndex
: This property is of type **long**.

type
: This property is of type **String**.

value
: This property is of type **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 of type **HTMLFormElement**.

accessKey
: This property is of type **String**.

htmlFor
: This property is of type **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 of type **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 of type **HTMLFormElement**.

accessKey
: This property is of type **String**.

align
: This property is of type **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 of type **boolean**.
type
This property is of type **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 of type **boolean**.
start
This property is of type **long**.
type
This property is of type **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 of type **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 of type **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 of type **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 of type **String**.
value
This property is of type **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 of type **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 of type **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 of type **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 of type **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 of type **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 of type **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 of type **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 of type **String**.

face

This property is of type **String**.

size

This property is of type **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 of type **String**.

face

This property is of type **String**.

size

This property is of type **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 of type **String**.

noShade

This property is of type **boolean**.

size

This property is of type **String**.

width

This property is of type **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 of type **String**.

dateTime

This property is of type **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 of type **String**.

charset

This property is of type **String**.

coords

This property is of type **String**.

href

This property is of type **String**.

hreflang

This property is of type **String**.

name

This property is of type **String**.

rel
This property is of type **String**.
rev
This property is of type **String**.
shape
This property is of type **String**.
tabIndex
This property is of type **long**.
target
This property is of type **String**.
type
This property is of type **String**.
blur
This property is of type **void**.
focus
This property is of type **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 of type **String**.
name
This property is of type **String**.
align
This property is of type **String**.
alt
This property is of type **String**.
border
This property is of type **String**.
height
This property is of type **String**.
hspace
This property is of type **String**.
isMap
This property is of type **boolean**.
longDesc
This property is of type **String**.
src
This property is of type **String**.
useMap
This property is of type **String**.
vspace
This property is of type **String**.
width
This property is of type **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 of type **HTMLFormElement**.

code

This property is of type **String**.

align

This property is of type **String**.

archive

This property is of type **String**.

border

This property is of type **String**.

codeBase

This property is of type **String**.

codeType

This property is of type **String**.

data

This property is of type **String**.

declare

This property is of type **boolean**.

height

This property is of type **String**.

hspace

This property is of type **String**.

name

This property is of type **String**.

standby

This property is of type **String**.

tabIndex

This property is of type **long**.

type

This property is of type **String**.

useMap

This property is of type **String**.

vspace

This property is of type **String**.

width

This property is of type **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 of type **String**.

type
This property is of type **String**.
value
This property is of type **String**.
valueType
This property is of type **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 of type **String**.
alt
This property is of type **String**.
archive
This property is of type **String**.
code
This property is of type **String**.
codeBase
This property is of type **String**.
height
This property is of type **String**.
hspace
This property is of type **String**.
name
This property is of type **String**.
object
This property is of type **String**.
vspace
This property is of type **String**.
width
This property is of type **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 of type **HTMLCollection**.
name
This property is of type **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 of type **String**.

alt
This property is of type **String**.
coords
This property is of type **String**.
href
This property is of type **String**.
noHref
This property is of type **boolean**.
shape
This property is of type **String**.
tabIndex
This property is of type **long**.
target
This property is of type **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 of type **String**.
htmlFor
This property is of type **String**.
event
This property is of type **String**.
charset
This property is of type **String**.
defer
This property is of type **boolean**.
src
This property is of type **String**.
type
This property is of type **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 of type **HTMLTableCaptionElement**.
tHead
This property is of type **HTMLTableSectionElement**.
tFoot
This property is of type **HTMLTableSectionElement**.
rows
This property is of type **HTMLCollection**.
tBodies
This property is of type **HTMLCollection**.

align
This property is of type **String**.
bgColor
This property is of type **String**.
border
This property is of type **String**.
cellPadding
This property is of type **String**.
cellSpacing
This property is of type **String**.
frame
This property is of type **String**.
rules
This property is of type **String**.
summary
This property is of type **String**.
width
This property is of type **String**.
createTHead
This property is of type **HTMLElement**.
deleteTHead
This property is of type **void**.
createTFoot
This property is of type **HTMLElement**.
deleteTFoot
This property is of type **void**.
createCaption
This property is of type **HTMLElement**.
deleteCaption
This property is of type **void**.

The **HTMLTableElement** object has the following methods:

insertRow(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteRow(index)
This method returns a **void**. The **index** parameter is 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 of type **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 of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
span
This property is of type **long**.
vAlign
This property is of type **String**.
width
This property is of type **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 of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
vAlign
This property is of type **String**.
rows
This property is of type **HTMLCollection**.

The **HTMLTableSectionElement** object has the following methods:

insertRow(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteRow(index)
This method returns a **void**. The **index** parameter is 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 of type **long**.
sectionRowIndex
This property is of type **long**.
cells
This property is of type **HTMLCollection**.
align
This property is of type **String**.
bgColor
This property is of type **String**.

ch
This property is of type **String**.
chOff
This property is of type **String**.
vAlign
This property is of type **String**.

The **HTMLTableRowElement** object has the following methods:

insertCell(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteCell(index)
This method returns a **void**. The **index** parameter is 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 of type **long**.
abbr
This property is of type **String**.
align
This property is of type **String**.
axis
This property is of type **String**.
bgColor
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
colSpan
This property is of type **long**.
headers
This property is of type **String**.
height
This property is of type **String**.
noWrap
This property is of type **boolean**.
rowSpan
This property is of type **long**.
scope
This property is of type **String**.
vAlign
This property is of type **String**.
width
This property is of type **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 of type **String**.

rows

This property is of type **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 of type **String**.

longDesc

This property is of type **String**.

marginHeight

This property is of type **String**.

marginWidth

This property is of type **String**.

name

This property is of type **String**.

noResize

This property is of type **boolean**.

scrolling

This property is of type **String**.

src

This property is of type **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 of type **String**.

frameBorder

This property is of type **String**.

height

This property is of type **String**.

longDesc

This property is of type **String**.

marginHeight

This property is of type **String**.

marginWidth

This property is of type **String**.

name

This property is of type **String**.

scrolling
This property is of type **String**.

src
This property is of type **String**.

width
This property is of type **String**.

Index

ATTRIBUTE_NODE 26
Attribute 37
CDATASection 44
CDATA_SECTION_NODE 26
COMMENT_NODE 26
CharacterData 34
Comment 43
DOCUMENT_FRAGMENT_NODE 26
DOCUMENT_NODE 26
DOCUMENT_TYPE_NODE 26
DOMImplementation 20
Document 21
DocumentFragment 21
DocumentType 44
ELEMENT_NODE 26
ENTITY_NODE 26
ENTITY_REFERENCE_NODE 26
Element 38
Entity 45
EntityReference 46
HTMLAnchorElement 79
HTMLAppletElement 85
HTMLAreaElement 87
HTMLBRElement 77
HTMLBaseElement 60
HTMLBaseFontElement 77
HTMLBlockquoteElement 75
HTMLBodyElement 61
HTMLButtonElement 71
HTMLCollection 51
HTMLDListElement 74
HTMLDirectoryElement 74
HTMLDivElement 76
HTMLDocument 52
HTMLElement 56
HTMLFieldSetElement 72
HTMLFontElement 78
HTMLFormElement 62
HTMLFrameElement 97
HTMLFrameSetElement 97
HTMLHRElement 78
HTMLHeadElement 57
HTMLHeadingElement 76
HTMLHtmlElement 57
HTMLIFrameElement 98
HTMLImageElement 81
HTMLInputElement 66
HTMLIsIndexElement 60
HTMLLIElement 75
HTMLLabelElement 72
HTMLLegendElement 73
HTMLLinkElement 58
HTMLMapElement 86
HTMLMenuElement 75
HTMLMetaElement 59
HTMLModElement 79
HTMLOListElement 73
HTMLObjectElement 82
HTMLOptGroupElement 65
HTMLOptionElement 65
HTMLParagraphElement 76
HTMLParamElement 84
HTMLPreElement 77
HTMLQuoteElement 76
HTMLScriptElement 88
HTMLSelectElement 63
HTMLStyleElement 60
HTMLTableCaptionElement 91
HTMLTableCellElement 95
HTMLTableColElement 92
HTMLTableElement 88
HTMLTableRowElement 94
HTMLTableSectionElement 93
HTMLTextAreaElement 69
HTMLTitleElement 59
HTMLUListElement 73
NOTATION_NODE 26
NamedNodeMap 32

Node 25
NodeList 31
Notation 45
PROCESSING_INSTRUCTION_NODE 26
ProcessingInstruction 43
TEXT_NODE 26
Text 42
URL 53
aLink 61
abbr 96
accept 67
acceptCharset 62
accessKey 67, 70, 71, 72, 73, 80, 87
action 62
add 64
align 67, 73, 76, 76, 76, 79, 81, 83, 85, 89, 92, 92, 93, 94, 96, 98
alt 68, 82, 85, 87
anchors 53
appendChild 30
appendData 35
applets 53
archive 83, 85
areas 86
attributes 28
axis 96
background 61
bgColor 61, 89, 94, 96
blur 65, 69, 70, 81
body 53
border 82, 83, 89
caption 89
cellIndex 96
cellPadding 90
cellSpacing 90
cells 94
ch 92, 93, 94, 96
chOff 92, 93, 94, 96
charset 58, 80, 88
checked 68
childNodes 28
cite 75, 77, 79
className 57
clear 77
click 69
cloneNode 30
close 54
code 83, 86
codeBase 83, 86
codeType 83
colSpan 96
color 78, 78
cols 70, 97
compact 73, 74, 74, 74, 75
content 59
cookie 54
coords 80, 87
createAttribute 24
createCDATASection 23
createCaption 91
createComment 23
createDocumentFragment 23
createElement 22
createEntityReference 24
createProcessingInstruction 24
createTFoot 90
createTHead 90
createTextNode 23
data 34, 43, 83
dateTime 79
declare 83
defaultChecked 67
defaultSelected 66
defaultValue 67, 70
defer 88
deleteCaption 91
deleteCell 95
deleteData 36
deleteRow 91, 93
deleteTFoot 91
deleteTHead 90
dir 57
disabled 58, 61, 64, 65, 66, 68, 70, 71
doctype 22
documentElement 22
domain 53
elements 62
enctype 62

entities 44
event 88
face 78, 78
firstChild 28
focus 65, 69, 71, 81
form 60, 64, 66, 67, 70, 71, 72, 72, 73, 83
forms 53
frame 90
frameBorder 97, 98
getAttribute 39
getAttributeNode 40
getElementById 55
getElementsByName 55
getElementsByTagName 25, 41
getNamedItem 32
hasChildNodes 30
hasFeature 20
headers 96
height 82, 84, 86, 96, 98
href 58, 60, 80, 87
hreflang 58, 80
hspace 82, 84, 86
htmlFor 72, 88
httpEquiv 59
id 57
images 53
implementation 22
index 66
insertBefore 28
insertCell 95
insertData 35
insertRow 91, 93
isMap 82
item 31, 33, 51
label 65, 66
lang 57
lastChild 28
length 31, 34, 35, 51, 62, 64
link 61
links 53
longDesc 82, 97, 99
lowSrc 81
marginHeight 98, 99
marginWidth 98, 99
maxLength 68
media 58, 61
method 62
multiple 64
name 37, 44, 59, 62, 64, 68, 70, 71, 80, 81, 84, 85, 86, 86, 98, 99
namedItem 52
nextSibling 28
noHref 87
noResize 98
noShade 79
noWrap 96
nodeName 27
nodeType 27
nodeValue 27
normalize 42
notationName 46
notations 45
object 86
open 54
options 64
ownerDocument 28
parentNode 27
previousSibling 28
profile 58
prompt 60
publicId 45, 46
readOnly 68, 70
referrer 53
rel 58, 80
remove 65
removeAttribute 40
removeAttributeNode 41
removeChild 29
removeNamedItem 33
replaceChild 29
replaceData 36
reset 63
rev 59, 80
rowIndex 94
rowSpan 96
rows 70, 89, 93, 97
rules 90
scheme 59

scope 96
scrolling 98, 99
sectionRowIndex 94
select 69, 71
selected 66
selectedIndex 63
setAttribute 39
setAttributeNode 40
setNamedItem 32
shape 80, 87
size 64, 68, 78, 78, 79
span 92
specified 38
splitText 42
src 68, 82, 88, 98, 99
standby 84
start 74
submit 63
substringData 35
summary 90
systemId 45, 46
tBodies 89
tFoot 89
tHead 89
tabIndex 64, 68, 70, 71, 80, 84, 87
tagName 39
target 43, 59, 60, 63, 80, 87
text 59, 61, 66, 88
title 53, 57
type 59, 61, 63, 68, 69, 72, 73, 74, 75, 81, 84, 85, 88
useMap 68, 82, 84
vAlign 92, 93, 95, 96
vLink 62
value 38, 64, 66, 68, 72, 75, 85
valueType 85
version 57
vspace 82, 84, 86
width 77, 79, 82, 84, 86, 90, 92, 97, 99
write 54
writeln 54

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 from a single set of XML source files. This section outlines how this specification is written in XML, and how the various derived works are created.

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 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.

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 pre-order 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 publicly 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.

The complete XML source files are available at:
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/xml-source.zip

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.