Document Object Model (DOM) Level 1 Specification (Second Edition)

Version 1.0

W3C Working Draft *29 September, 2000*

This version:
http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929
(PostScript , PDF file , plain text , ZIP file)
Latest version:
http://www.w3.org/TR/REC-DOM-Level-1
Previous version:
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001

Editors:
Lauren Wood, *SoftQuad, Inc., chair*
Arnaud Le Hors, *W3C, staff contact*
Vidur Apparao, *Netscape*
Steve Byrne, *Sun*
Mike Champion, *ArborText*
Scott Isaacs, *Microsoft*
Ian Jacobs, *W3C*
Gavin Nicol, *Inso EPS*
Jonathan Robie, *Texcel Research*
Robert Sutor, *IBM*
Chris Wilson, *Microsoft*

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.

Status of this document

This document is a version of the DOM Level 1 Recommendation incorporating the errata changes as of September 29, 2000. It is released by the DOM Working Group as a W3C Working Draft to gather public feedback before its final release as the DOM Level 1 second edition W3C Recommendation (as these changes are editorials, there will be no Candidate Recommendation or Proposed Recommendation stages). The review period for this Working Draft is 4 weeks ending October 27 2000.

This second edition is not a new version of the DOM Level 1; it merely incorporates the changes dictated by the first-edition errata list. This document should not be used as reference material or cited as a normative reference from another document.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM WG members. Different modules of the Document Object Model have different editors.

Please report errors in this document to the public mailing list www-dom@w3.org. An archive is available at http://lists.w3.org/Archives/Public/www-dom/.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Table of contents

Table of contents

Expanded Table of Contents

Copyright Notice

This document is published under the W3C Document Copyright Notice and License [p.7] . The bindings within this document are published under the W3C Software Copyright Notice and License [p.8] . The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL Definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java Language binding, the package names can no longer be in the 'org.w3c' package.

W3C Document Copyright Notice and License

Note: This section is a copy of the W3C Document Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-documents-19990405.

http://www.w3.org/Consortium/Legal/

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) 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 URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/" (Hypertext is preferred, but a textual representation is permitted.)
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in 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 of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

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.

W3C Software Copyright Notice and License

Note: This section is a copy of the W3C Software Copyright Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and modify this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers. If none exist, then a notice of the following form: "Copyright © [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/."
3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We

recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION 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 SOFTWARE OR DOCUMENTATION.

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

What is the Document Object Model?

Editors
Jonathan Robie, Texcel Research

Introduction

The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed 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 Object Management Group (OMG) IDL [OMGIDL], as defined in the CORBA 2.2 specification [CORBA]. In addition to the OMG IDL specification, we provide language bindings for Java [Java] and ECMAScript [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.

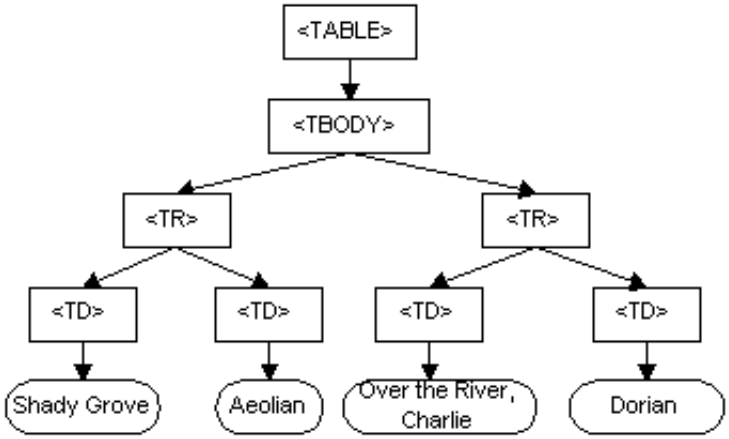
What the Document Object Model is

The DOM is a programming API for documents. It is based on an object structure that 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, which is like a "forest" or "grove", which can contain more than one tree. Each document contains zero or one doctype nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document. 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 also use the term "tree" when referring to the arrangement of those information items which can be reached by using "tree-walking" methods; (this does not include attributes). 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, in accordance with the XML Information Set [Infoset].

Note: There may be some variations depending on the parser being used to build the DOM. For instance, the DOM may not contain whitespaces in element content if the parser discards them.

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.

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 binding 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 what information in a document is relevant or how information in a document is structured. For XML, this is specified by the W3C XML Information Set [Infoset]. The DOM is simply an API to this information set.
- 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 API 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. Since numeric character references and pre-defined entities are not recognized as such in CDATA sections, or in the SCRIPT and STYLE elements in HTML, they are not replaced by the single character they appear to refer to. If the example above were enclosed in a CDATA section, the "&" would not be replaced by "&"; neither would the <p> be recognized as a start tag. 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. Also, implementations should be aware of the fact that serialization into a character encoding ("charset") that does not fully cover ISO 10646 may fail if there are characters in markup or CDATA sections that are not present in the encoding.

Compliance

The Document Object Model Level 1 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.

A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.22] interface to determine whether the module is supported or not. The feature strings for all modules in DOM Level 1 are listed in the following table; (strings are case-insensitive):

Module	Feature String
XML	XML
HTML	HTML

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 attributes 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 cannot know what constructors to call for an implementation. In general, DOM users call the createX() 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 createX() functions.

Limitations of Level 1

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

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 set of objects and interfaces for accessing and manipulating document objects. The functionality specified in this section (the *Core* functionality) is 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 creation and population of a `Document` [p.23] object using only DOM API calls; loading a `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` [p.28] 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. For XML and HTML, the node types, and which node types they may have as children, are as follows:

- `Document` [p.23] -- `Element` [p.43] (maximum of one), `ProcessingInstruction` [p.52] , `Comment` [p.48] , `DocumentType` [p.49] (maximum of one)
- `DocumentFragment` [p.23] -- `Element` [p.43] , `ProcessingInstruction` [p.52] , `Comment` [p.48] , `Text` [p.47] , `CDATASection` [p.48] , `EntityReference` [p.52]
- `DocumentType` [p.49] -- no children
- `EntityReference` [p.52] -- `Element` [p.43] , `ProcessingInstruction` [p.52] , `Comment` [p.48] , `Text` [p.47] , `CDATASection` [p.48] , `EntityReference`
- `Element` [p.43] -- `Element`, `Text` [p.47] , `Comment` [p.48] , `ProcessingInstruction` [p.52] , `CDATASection` [p.48] , `EntityReference` [p.52]
- `Attr` [p.42] -- `Text` [p.47] , `EntityReference` [p.52]
- `ProcessingInstruction` [p.52] -- no children
- `Comment` [p.48] -- no children
- `Text` [p.47] -- no children
- `CDATASection` [p.48] -- no children
- `Entity` [p.51] -- `Element` [p.43] , `ProcessingInstruction` [p.52] , `Comment` [p.48] , `Text` [p.47] , `CDATASection` [p.48] , `EntityReference` [p.52]
- `Notation` [p.50] -- no children

The DOM also specifies a `NodeList` [p.35] interface to handle ordered lists of `Nodes` [p.28] , such as the children of a `Node` [p.28] , or the elements returned by the `getElementsByTagName` method of the `Element` [p.43] interface, and also a `NamedNodeMap` [p.36] interface to handle unordered sets of

nodes referenced by their name attribute, such as the attributes of an `Element`. `NodeList` [p.35] and `NamedNodeMap` [p.36] objects in the DOM are *live*; that is, changes to the underlying document structure are reflected in all relevant `NodeList` and `NamedNodeMap` objects. For example, if a DOM user gets a `NodeList` object containing the children of an `Element` [p.43] , 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` [p.28] in the tree are reflected in all references to that `Node` in `NodeList` and `NamedNodeMap` objects.

Finally, the interfaces `Text` [p.47] , `Comment` [p.48] , and `CDATASection` [p.48] all inherit from the `CharacterData` [p.38] interface.

1.1.2. Memory Management

Most of the APIs defined by this specification are *interfaces* rather than classes. That means that an implementation need only expose methods with the defined names and specified operation, not 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` [p.23] 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` [p.22] or `Document` [p.23] objects; DOM implementations must provide some proprietary way of bootstrapping these DOM interfaces, and then all other objects can be built from there.

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 bindings that 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++) may require such support. These extensions will be the responsibility of those adapting the DOM API to a specific language, not the DOM Working Group.

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 make it difficult to avoid naming conflicts with short, familiar names. So, some 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` [p.28] 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` [p.28] 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` [p.43] 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 `DOMString` type

To ensure interoperability, the DOM specifies the following:

- **Type Definition *DOMString***

  A `DOMString` [p.19] is a sequence of *16-bit units* [p.125] .
  IDL Definition

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

- Applications must encode `DOMString` [p.19] using UTF-16 (defined in [Unicode] and Amendment 1 of [ISO/IEC 10646]).
  The UTF-16 encoding was chosen because of its widespread industry practice. Note that for both HTML and XML, the document character set (and therefore the notation of numeric character references) is based on UCS [ISO-10646]. A single numeric character reference in a source document may therefore in some cases correspond to two 16-bit units in a `DOMString` [p.19] (a high surrogate and a low surrogate).

  Note: Even though the DOM defines the name of the string type to be `DOMString` [p.19] , bindings may use different names. For example for Java, `DOMString` 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 negotiation to decide the width and encoding of a character.

1.1.6. String comparisons 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 is performed purely by binary comparison of the *16-bit units* [p.125] of the `DOMString` [p.19] . In addition, the DOM assumes that any case normalizations take place in the processor, *before* the DOM structures are built.

Note: Besides case folding, there are additional normalizations that can be applied to text. The W3C I18N Working Group is in the process of defining exactly which normalizations are necessary, and where they should be applied. The W3C I18N Working Group expects to require early normalization, which means that data read into the DOM is assumed to already be normalized. The DOM and applications built on top of it in this case only have to assure that text remains normalized when being changed. For further details, please see [Charmod].

1.2. Fundamental Interfaces

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

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 situations, such as out-of-bound errors when using `NodeList` [p.35] .

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 {
  unsigned short   code;
};
// ExceptionCode
const unsigned short      INDEX_SIZE_ERR                = 1;
const unsigned short      DOMSTRING_SIZE_ERR            = 2;
const unsigned short      HIERARCHY_REQUEST_ERR         = 3;
const unsigned short      WRONG_DOCUMENT_ERR            = 4;
const unsigned short      INVALID_CHARACTER_ERR         = 5;
const unsigned short      NO_DATA_ALLOWED_ERR           = 6;
const unsigned short      NO_MODIFICATION_ALLOWED_ERR   = 7;
const unsigned short      NOT_FOUND_ERR                 = 8;
const unsigned short      NOT_SUPPORTED_ERR             = 9;
const unsigned short      INUSE_ATTRIBUTE_ERR           = 10;
```

Definition group *ExceptionCode*

An integer indicating the type of error generated.

Note: Other numeric codes are reserved for W3C for possible future use.

Defined Constants

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

`HIERARCHY_REQUEST_ERR`
: If any node is inserted somewhere it doesn’t belong

`INDEX_SIZE_ERR`
: If index or size is negative, or greater than the allowed value

`INUSE_ATTRIBUTE_ERR`
: If an attempt is made to add an attribute that is already in use elsewhere

`INVALID_CHARACTER_ERR`
: If an invalid or illegal character is specified, such as in a name. See *production 2* in the XML specification for the definition of a legal character, and *production 5* for the definition of a legal name character.

`NOT_FOUND_ERR`
: If an attempt is 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

`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

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

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 DOMString feature,
                                in DOMString version);
};
```

Methods

`hasFeature`
Test if the DOM implementation implements a specific feature.

Parameters

`feature` of type `DOMString` [p.19]
The name of the feature to test (case-insensitive). The values used by DOM features are defined throughout this specification and listed in the Compliance [p.14] section. The name must be an *XML name* [p.128] . To avoid possible conflicts, as a convention, names referring to features defined outside the DOM specification should be made unique by reversing the name of the Internet domain name of the person (or the organization that the person belongs to) who defines the feature, component by component, and using this as a prefix. For instance, the W3C SYMM Working Group defines the feature "org.w3c.dom.smil".

`version` of type `DOMString`
This is the version number of the feature to test. In Level 1, this is the string "1.0". If the version is not specified, supporting any version of the feature causes the method to return `true`.

Return Value

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

No Exceptions

Interface *DocumentFragment*

`DocumentFragment` is a "lightweight" or "minimal" `Document` [p.23] object. It is very common to want to be able to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments and it is quite natural to use a Node for this purpose. While it is true that a `Document` object could fulfill this role, a `Document` object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. `DocumentFragment` is such an object.

Furthermore, various operations -- such as inserting nodes as children of another `Node` [p.28] -- 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` nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a `DocumentFragment` might have only one child and that child node could be a `Text` [p.47] node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a `DocumentFragment` is inserted into a `Document` [p.23] (or indeed any other `Node` [p.28] 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` [p.28] 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 DOMString tagName)
                                       raises(DOMException);
  DocumentFragment   createDocumentFragment();
  Text               createTextNode(in DOMString data);
  Comment            createComment(in DOMString data);
  CDATASection       createCDATASection(in DOMString data)
                                       raises(DOMException);
  ProcessingInstruction createProcessingInstruction(in DOMString target,
                                                     in DOMString data)
                                       raises(DOMException);
  Attr               createAttribute(in DOMString name)
                                       raises(DOMException);
  EntityReference    createEntityReference(in DOMString name)
                                       raises(DOMException);
  NodeList           getElementsByTagName(in DOMString tagname);
};
```

Attributes

`doctype` of type `DocumentType` [p.49] , readonly

The Document Type Declaration (see `DocumentType` [p.49]) associated with this document. For HTML documents as well as XML documents without a document type declaration this returns `null`. The DOM Level 1 does not support editing the Document Type Declaration. `docType` cannot be altered in any way, including through the use of methods inherited from the `Node` [p.28] interface, such as `insertNode` or `removeNode`.

`documentElement` of type `Element` [p.43] , readonly

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".

`implementation` of type `DOMImplementation` [p.22] , readonly

The `DOMImplementation` [p.22] object that handles this document. A DOM application may use objects from multiple implementations.

Methods

`createAttribute`

Creates an `Attr` [p.42] of the given name. Note that the `Attr` instance can then be set on an `Element` [p.43] using the `setAttributeNode` method.

Parameters

`name` of type `DOMString` [p.19]

The name of the attribute.

Return Value

`Attr` [p.42]	A new `Attr` object with the `NodeName` attribute set to `name`. The value of the attribute is the empty string.

Exceptions

`DOMException` [p.20]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createCDATASection`

Creates a `CDATASection` [p.48] node whose value is the specified string.

Parameters

`data` of type `DOMString` [p.19]

The data for the `CDATASection` [p.48] contents.

Return Value

`CDATASection` [p.48]	The new `CDATASection` object.

Exceptions

`DOMException` [p.20]	NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`createComment`

Creates a `Comment` [p.48] node given the specified string.

Parameters

`data` of type `DOMString` [p.19]

The data for the node.

Return Value

`Comment` [p.48]	The new `Comment` object.

No Exceptions

`createDocumentFragment`

Creates an empty `DocumentFragment` [p.23] object.

Return Value

`DocumentFragment` [p.23]	A new `DocumentFragment`.

No Parameters
No Exceptions

`createElement`

Creates an element of the type specified. Note that the instance returned implements the `Element` [p.43] interface, so attributes can be specified directly on the returned object. In addition, if there are known attributes with default values, `Attr` [p.42] nodes representing them are automatically created and attached to the element.

Parameters

`tagName` of type `DOMString` [p.19]

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 Value

`Element` [p.43]	A new `Element` object with the `nodeName` attribute set to `tagName`.

Exceptions

`DOMException` [p.20]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createEntityReference`

Creates an `EntityReference` [p.52] object. In addition, if the referenced entity is known, the child list of the `EntityReference` node is made the same as that of the corresponding `Entity` [p.51] node.

Parameters

`name` of type `DOMString` [p.19]

The name of the entity to reference.

Return Value

`EntityReference` [p.52]	The new `EntityReference` object.

Exceptions

`DOMException` [p.20]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character. NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`createProcessingInstruction`

Creates a `ProcessingInstruction` [p.52] node given the specified name and data strings.

Parameters

`target` of type `DOMString` [p.19]

The target part of the processing instruction.

`data` of type `DOMString`

The data for the node.

Return Value

`ProcessingInstruction` [p.52]	The new `ProcessingInstruction` object.

Exceptions

`DOMException` [p.20]	INVALID_CHARACTER_ERR: Raised if the specified target contains an illegal character. NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`createTextNode`

Creates a `Text` [p.47] node given the specified string.

Parameters

`data` of type `DOMString` [p.19]

The data for the node.

Return Value

`Text` [p.47]	The new `Text` object.

No Exceptions

`getElementsByTagName`

Returns a `NodeList` [p.35] of all the `Elements` [p.43] with a given tag name in the order in which they are encountered in a preorder traversal of the `Document` tree.

Parameters

`tagname` of type `DOMString` [p.19]

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

Return Value

`NodeList` [p.35]	A new `NodeList` object containing all the matched `Elements` [p.43] .

No Exceptions

Interface *Node*

The `Node` interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the `Node` interface expose methods for dealing with children, not all objects implementing the `Node` interface may have children. For example, `Text` [p.47] nodes may not have children, and adding children to such nodes results in a `DOMException` [p.20] 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` [p.48]), 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 DOMString        nodeName;
           attribute DOMString        nodeValue;
                                        // raises(DOMException) on setting
                                        // raises(DOMException) on retrieval

  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)
                                         raises(DOMException);
};
```

Definition group *NodeType*

An integer indicating which type of node this is.

Note: Numeric codes up to 200 are reserved to W3C for possible future use.

Defined Constants

`ATTRIBUTE_NODE`
: The node is an `Attr` [p.42] .

`CDATA_SECTION_NODE`
: The node is a `CDATASection` [p.48] .

`COMMENT_NODE`
: The node is a `Comment` [p.48] .

`DOCUMENT_FRAGMENT_NODE`
: The node is a `DocumentFragment` [p.23] .

`DOCUMENT_NODE`
: The node is a `Document` [p.23] .

`DOCUMENT_TYPE_NODE`
: The node is a `DocumentType` [p.49] .

`ELEMENT_NODE`
: The node is an `Element` [p.43] .

`ENTITY_NODE`
: The node is an `Entity` [p.51] .

`ENTITY_REFERENCE_NODE`
: The node is an `EntityReference` [p.52] .

`NOTATION_NODE`
: The node is a `Notation` [p.50] .

`PROCESSING_INSTRUCTION_NODE`
The node is a `ProcessingInstruction` [p.52] .

`TEXT_NODE`
The node is a `Text` [p.47] node.

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

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

Attributes

`attributes` of type `NamedNodeMap` [p.36] , readonly
A `NamedNodeMap` [p.36] containing the attributes of this node (if it is an `Element` [p.43]) or `null` otherwise.

`childNodes` of type `NodeList` [p.35] , readonly
A `NodeList` [p.35] that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes.

`firstChild` of type `Node` [p.28] , readonly
The first child of this node. If there is no such node, this returns `null`.

`lastChild` of type `Node` [p.28] , readonly
The last child of this node. If there is no such node, this returns `null`.

`nextSibling` of type `Node` [p.28] , readonly
The node immediately following this node. If there is no such node, this returns `null`.

`nodeName` of type `DOMString` [p.19] , readonly
The name of this node, depending on its type; see the table above.

`nodeType` of type `unsigned short`, readonly
A code representing the type of the underlying object, as defined above.

`nodeValue` of type `DOMString` [p.19]
The value of this node, depending on its type; see the table above. When it is defined to be `null`, setting it has no effect.
Exceptions on setting

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Exceptions on retrieval

`DOMException` [p.20]	DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` [p.19] variable on the implementation platform.

`ownerDocument` of type `Document` [p.23] , readonly
The `Document` [p.23] object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document`, this is `null`.

`parentNode` of type `Node` [p.28] , readonly
The parent of this node. All nodes, except `Attr` [p.42] , `Document` [p.23] , `DocumentFragment` [p.23] , `Entity` [p.51] , and `Notation` [p.50] 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`.

`previousSibling` of type `Node` [p.28] , readonly
The node immediately preceding this node. If there is no such node, this returns `null`.

Methods

`appendChild`
Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.
Parameters
`newChild` of type `Node` [p.28]
The node to add.
If it is a `DocumentFragment` [p.23] object, the entire contents of the document

fragment are moved into the child list of this node

Return Value

`Node` [p.28]	The node added.

Exceptions

`DOMException` [p.20]	HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node's ancestors. 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.

`cloneNode`

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent; (`parentNode` is `null`.).
Cloning an `Element` [p.43] 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` [p.47] node. Cloning an `Attribute` directly, as opposed to be cloned as part of an `Element` cloning operation, returns a specified attribute (`specified` is `true`). Cloning any other type of node simply returns a copy of this node.
Note that cloning an immutable subtree results in a mutable copy, but the children of an `EntityReference` [p.52] clone are *readonly* [p.128] . In addition, clones of unspecified `Attr` [p.42] nodes are specified. And, cloning `Document` [p.23] , `DocumentType` [p.49] , `Entity` [p.51] , and `Notation` [p.50] nodes is implementation dependent.
Parameters
`deep` of type `boolean`

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` [p.43]).

Return Value

`Node` [p.28]	The duplicate node.

Exceptions

`DOMException` [p.20]	NOT_SUPPORTED_ERR: Raised if this node is a of type `DOCUMENT_NODE`, `DOCUMENT_TYPE_NODE`, `ENTITY_NODE`, or `NOTATION_NODE` and the implementation does not support cloning this type of node.

`hasChildNodes`

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

Return Value

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

No Parameters

No Exceptions

`insertBefore`

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

If `newChild` is a `DocumentFragment` [p.23] 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` of type `Node` [p.28]

The node to insert.

`refChild` of type `Node`

The reference node, i.e., the node before which the new node must be inserted.

Return Value

`Node` [p.28]	The node being inserted.

Exceptions

`DOMException` [p.20]	HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors. WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly or if the parent of the node being inserted is readonly. NOT_FOUND_ERR: Raised if `refChild` is not a child of this node.

`removeChild`

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

Parameters

`oldChild` of type `Node` [p.28]

The node being removed.

Return Value

`Node` [p.28]	The node removed.

Exceptions

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly. NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.

`replaceChild`

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node.

If `newChild` is a `DocumentFragment` [p.23] object, `oldChild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newChild` is already in the tree, it is first removed.

Parameters

`newChild` of type `Node` [p.28]

The new node to put in the child list.

`oldChild` of type `Node`
The node being replaced in the list.

Return Value

`Node` [p.28]	The node replaced.

Exceptions

`DOMException` [p.20]	HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or it the node to put in is one of this node's ancestors. WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node. NO_MODIFICATION_ALLOWED_ERR: Raised if this node or the parent of the new node is readonly. NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.

Interface *NodeList*

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. `NodeList` objects in the DOM are *live* [p.18] .

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;
};
```

Attributes

`length` of type `unsigned long`, readonly
The number of nodes in the list. The range of valid child node indices is 0 to `length-1` inclusive.

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, this returns `null`.

Parameters

`index` of type `unsigned long`
Index into the collection.

Return Value

`Node` [p.28]	The node at the `index`th position in the `NodeList`, or `null` if that is not a valid index.

No Exceptions

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` [p.35] ; `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.

IDL Definition

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

`NamedNodeMap` objects in the DOM are *live* [p.18] .

Attributes

`length` of type `unsigned long`, readonly
The number of nodes in this map. The range of valid child node indices is `0` to `length-1` inclusive.

Methods

`getNamedItem`
Retrieves a node specified by name.

Parameters

`name` of type `DOMString` [p.19]
The `nodeName` of a node to retrieve.

Return Value

`Node` [p.28]	A `Node` (of any type) with the specified `nodeName`, or `null` if it does not identify any node in this map.

No Exceptions

`item`

Returns the `index`th item in the map. If `index` is greater than or equal to the number of nodes in this map, this returns `null`.

Parameters

`index` of type `unsigned long`

Index into this map.

Return Value

`Node` [p.28]	The node at the `index`th position in the map, or `null` if that is not a valid index.

No Exceptions

`removeNamedItem`

Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value.

Parameters

`name` of type `DOMString` [p.19]

The `nodeName` of the node to remove.

Return Value

`Node` [p.28]	The node removed from this map if a node with such a name exists.

Exceptions

`DOMException` [p.20]	NOT_FOUND_ERR: Raised if there is no node named `name` in this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`setNamedItem`

Adds a node using its `nodeName` attribute. If a node with that name is already present in this map, it is replaced by the new one.

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` of type `Node` [p.28]

A node to store in this map. The node will later be accessible using the value of its `nodeName` attribute.

Return Value

`Node` [p.28]	If the new `Node` replaces an existing node the replaced `Node` is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.20]	WRONG_DOCUMENT_ERR: Raised if `arg` was created from a different document than the one that created this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly. INUSE_ATTRIBUTE_ERR: Raised if `arg` is an `Attr` [p.42] that is already an attribute of another `Element` [p.43] object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.

Interface *CharacterData*

The `CharacterData` interface extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to `CharacterData`, though `Text` [p.47] and others do inherit the interface from it. All `offsets` in this interface start from `0`.

As explained in the `DOMString` [p.19] interface, text strings in the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In the following, the term *16-bit units* [p.125] is used whenever necessary to indicate that indexing on CharacterData is done in 16-bit units.

IDL Definition

```
interface CharacterData : Node {
           attribute DOMString        data;
                                        // raises(DOMException) on setting
                                        // raises(DOMException) on retrieval

  readonly attribute unsigned long    length;
  DOMString          substringData(in unsigned long offset,
                                   in unsigned long count)
                                        raises(DOMException);
  void               appendData(in DOMString arg)
                                        raises(DOMException);
  void               insertData(in unsigned long offset,
                                in DOMString 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 DOMString arg)
                                           raises(DOMException);
};
```

Attributes

`data` of type `DOMString` [p.19]

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a single `DOMString` [p.19] . In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

Exceptions on setting

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Exceptions on retrieval

`DOMException` [p.20]	DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` [p.19] variable on the implementation platform.

`length` of type `unsigned long`, readonly

The number of *16-bit units* [p.125] that are available through `data` and the `substringData` method below. This may have the value zero, i.e., `CharacterData` nodes may be empty.

Methods

`appendData`

Append the string to the end of the character data of the node. Upon success, `data` provides access to the concatenation of `data` and the `DOMString` [p.19] specified.

Parameters

`arg` of type `DOMString` [p.19]

The `DOMString` to append.

Exceptions

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`deleteData`

Remove a range of *16-bit units* [p.125] from the node. Upon success, `data` and `length` reflect the change.

Parameters

`offset` of type `unsigned long`

The offset from which to start removing.

`count` of type `unsigned long`

The number of 16-bit units to delete. If the sum of `offset` and `count` exceeds `length` then all 16-bit units from `offset` to the end of the data are deleted.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified `offset` is negative or greater than the number of 16-bit units in `data`, or if the specified `count` is negative. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`insertData`

Insert a string at the specified *16-bit unit* [p.125] offset.

Parameters

`offset` of type `unsigned long`

The character offset at which to insert.

`arg` of type `DOMString` [p.19]

The `DOMString` to insert.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified `offset` is negative or greater than the number of 16-bit units in `data`. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`replaceData`

Replace the characters starting at the specified *16-bit unit* [p.125] offset with the specified string.

Parameters

`offset` of type `unsigned long`

The offset from which to start replacing.

`count` of type `unsigned long`
The number of 16-bit units to replace. If the sum of `offset` and `count` exceeds `length`, then all 16-bit units to the end of the data are replaced; (i.e., the effect is the same as a `remove` method call with the same range, followed by an `append` method invocation).

`arg` of type `DOMString` [p.19]
The `DOMString` with which the range must be replaced.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified `offset` is negative or greater than the number of 16-bit units in `data`, or if the specified `count` is negative. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`substringData`
Extracts a range of data from the node.
Parameters
`offset` of type `unsigned long`
Start offset of substring to extract.

`count` of type `unsigned long`
The number of 16-bit units to extract.

Return Value

`DOMString` [p.19]	The specified substring. If the sum of `offset` and `count` exceeds the `length`, then all 16-bit units to the end of the data are returned.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified `offset` is negative or greater than the number of 16-bit units in `data`, or if the specified `count` is negative. DOMSTRING_SIZE_ERR: Raised if the specified range of text does not fit into a `DOMString` [p.19] .

Interface *Attr*

The `Attr` interface represents an attribute in an `Element` [p.43] object. Typically the allowable values for the attribute are defined in a document type definition.

`Attr` objects inherit the `Node` [p.28] interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the `Node` attributes `parentNode`, `previousSibling`, and `nextSibling` have a `null` value for `Attr` objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with; this should make it more efficient to implement such features as default attributes associated with all elements of a given type. Furthermore, `Attr` nodes may not be immediate children of a `DocumentFragment` [p.23] . However, they can be associated with `Element` [p.43] nodes contained within a `DocumentFragment`. In short, users and implementors of the DOM need to be aware that `Attr` nodes have some things in common with other objects inheriting the `Node` interface, but they also are quite distinct.

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

In XML, where the value of an attribute can contain entity references, the child nodes of the `Attr` node provide a representation in which entity references are not expanded. These child nodes may be either `Text` [p.47] or `EntityReference` [p.52] nodes. Because the attribute type may be unknown, there are no tokenized attribute values.

IDL Definition

```
interface Attr : Node {
  readonly attribute DOMString        name;
  readonly attribute boolean          specified;
  // Modified in DOM Level 1:
           attribute DOMString        value;
                                        // raises(DOMException) on setting

};
```

Attributes

`name` of type `DOMString` [p.19] , readonly

Returns the name of this attribute.

`specified` of type `boolean`, readonly

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. The implementation will then make a new attribute available with `specified` set to `false` and the default value (if one exists).
In summary:

- If the attribute has an assigned value in the document then `specified` is `true`, and 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.
- If the attribute is not associated to any element (i.e. because it was just created or was obtained from some removal or cloning operation) `specified` is `true`.

`value` of type `DOMString` [p.19] , modified in **DOM Level 1**

On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values. See also the method `getAttribute` on the `Element` [p.43] interface.
On setting, this creates a `Text` [p.47] node with the unparsed contents of the string. I.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `setAttribute` on the `Element` [p.43] interface.
Exceptions on setting

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Interface *Element*

The `Element` interface represents an element in an HTML or XML document. Elements may have attributes associated with them; since the `Element` interface inherits from `Node` [p.28] , the generic `Node` interface attribute `attributes` may be used to retrieve the set of all attributes for an element. There are methods on the `Element` interface to retrieve either an `Attr` [p.42] object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an `Attr` object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience.
IDL Definition

```
interface Element : Node {
  readonly attribute DOMString        tagName;
  DOMString          getAttribute(in DOMString name);
  void               setAttribute(in DOMString name,
                                  in DOMString value)
                                        raises(DOMException);
  void               removeAttribute(in DOMString name)
                                        raises(DOMException);
  Attr               getAttributeNode(in DOMString name);
  Attr               setAttributeNode(in Attr newAttr)
                                        raises(DOMException);
  Attr               removeAttributeNode(in Attr oldAttr)
```

```
                                        raises(DOMException);
  NodeList          getElementsByTagName(in DOMString name);
  void              normalize();
};
```

Attributes

`tagName` of type `DOMString` [p.19] , readonly

The name of the element. 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` of type `DOMString` [p.19]

The name of the attribute to retrieve.

Return Value

`DOMString` [p.19]	The `Attr` [p.42] value as a string, or the empty string if that attribute does not have a specified or default value.

No Exceptions

`getAttributeNode`

Retrieves an `Attr` [p.42] node by name.

Parameters

`name` of type `DOMString` [p.19]

The name of the attribute to retrieve.

Return Value

`Attr` [p.42]	The `Attr` node with the specified attribute name or `null` if there is no such attribute.

No Exceptions

`getElementsByTagName`

Returns a `NodeList` [p.35] of all descendant `Elements` with a given tag name, in the order in which they would be encountered in a preorder traversal of the `Element` tree.

Parameters

`name` of type `DOMString` [p.19]

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

Return Value

`NodeList` [p.35]	A list of matching `Element` nodes.

No Exceptions

`normalize`

Puts all `Text` [p.47] nodes in the full depth of the sub-tree underneath this `Element`, including attribute nodes, into a "normal" form where only markup (e.g., tags, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are no adjacent `Text` nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as XPointer [XPointer] lookups) that depend on a particular document tree structure are to be used.

Note: In cases where the document contains `CDATASections` [p.48] , the normalize operation alone may not be sufficient, since XPointers do not differentiate between `Text` [p.47] nodes and `CDATASection` [p.48] nodes.

No Parameters
No Return Value
No Exceptions

`removeAttribute`

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

Parameters

`name` of type `DOMString` [p.19]

The name of the attribute to remove.

Exceptions

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`removeAttributeNode`

Removes the specified attribute. If the removed `Attr` [p.42] has a default value it is immediately replaced.

Parameters

`oldAttr` of type `Attr` [p.42]
The `Attr` node to remove from the attribute list.

Return Value

`Attr` [p.42]	The `Attr` node that was removed.

Exceptions

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly. NOT_FOUND_ERR: Raised if `oldAttr` is not an attribute of the element.

`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 `Attr` [p.42] node plus any `Text` [p.47] and `EntityReference` [p.52] nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.
Parameters
`name` of type `DOMString` [p.19]
The name of the attribute to create or alter.

`value` of type `DOMString`
Value to set in string form.

Exceptions

`DOMException` [p.20]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

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

`newAttr` of type `Attr` [p.42]
The `Attr` node to add to the attribute list.

Return Value

`Attr` [p.42]	If the `newAttr` attribute replaces an existing attribute, the replaced `Attr` node is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.20]	WRONG_DOCUMENT_ERR: Raised if `newAttr` was created from a different document than the one that created the element. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly. INUSE_ATTRIBUTE_ERR: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` [p.42] nodes to re-use them in other elements.

Interface *Text*

The `Text` interface inherits from `CharacterData` [p.38] and represents the textual content (termed *character data* in XML) of an `Element` [p.43] or `Attr` [p.42] . If there is no markup inside an element's content, the text is contained in a single object implementing the `Text` interface that is the only child of the element. If there is markup, it is parsed into the *information items* [p.127] (elements, comments, etc.) and `Text` nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one `Text` node for each block of text. Users may create adjacent `Text` nodes that represent the contents of a given element without any intervening markup, but should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not (in general) persist between DOM editing sessions. The `normalize()` method on `Element` [p.43] merges any such adjacent `Text` objects into a single node for each block of text.

IDL Definition

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

Methods

`splitText`
Breaks this node into two nodes at the specified `offset`, keeping both in the tree as siblings. This node then only contains all the content up to the `offset` point. A new node

of the same type, which is inserted as the next sibling of this node, contains all the content at and after the `offset` point. When the `offset` is equal to the length of this node, the new node has no data.

Parameters

`offset` of type `unsigned long`

The *16-bit unit* [p.125] offset at which to split, starting from `0`.

Return Value

`Text` [p.47]	The new node, of the same type as this node.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of 16-bit units in `data`.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

Interface *Comment*

This interface inherits from `CharacterData` [p.38] and represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'. Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

IDL Definition

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

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.

A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.22] interface to determine whether they are supported or not. The feature string for all the interfaces listed in this section is "XML" and the version is "1.0".

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 recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections cannot be nested. Their primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The `DOMString` [p.19] attribute of the `Text` [p.47] 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 and that, depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The `CDATASection` interface inherits from the `CharacterData` [p.38] interface through the `Text` [p.47] interface. Adjacent `CDATASection` nodes are not merged by use of the normalize method on the `Element` [p.43] interface.

Note: Because no markup is recognized within a `CDATASection`, character numeric references cannot be used as an escape mechanism when serializing. Therefore, action needs to be taken when serializing a `CDATASection` with a character encoding where some of the contained characters cannot be represented. Failure to do so would not produce well-formed XML.
One potential solution in the serialization process is to end the CDATA section before the character, output the character using a character reference or entity reference, and open a new CDATA section for any further characters in the text node. Note, however, that some code conversion libraries at the time of writing do not return an error or exception when a character is missing from the encoding, making the task of ensuring that data is not corrupted on serialization more difficult.

IDL Definition

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

Interface *DocumentType*

Each `Document` [p.23] has a `doctype` attribute whose value is either `null` or a `DocumentType` object. The `DocumentType` interface 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.

The DOM Level 1 doesn’t support editing `DocumentType` nodes.
IDL Definition

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

Attributes

`entities` of type `NamedNodeMap` [p.36] , readonly

A `NamedNodeMap` [p.36] containing the general entities, both external and internal, declared in the DTD. Parameter entities are not contained. Duplicates are discarded. For example in:

```
<!DOCTYPE ex SYSTEM "ex.dtd" [
  <!ENTITY foo "foo">
  <!ENTITY bar "bar">
  <!ENTITY bar "bar2">
  <!ENTITY % baz "baz">
]>
<ex/>
```

the interface provides access to `foo` and the first declaration of `bar` but not the second declaration of `bar` or `baz`. Every node in this map also implements the `Entity` [p.51] interface.
The DOM Level 1 does not support editing entities, therefore `entities` cannot be altered in any way.

`name` of type `DOMString` [p.19] , readonly
The name of DTD; i.e., the name immediately following the `DOCTYPE` keyword.

`notations` of type `NamedNodeMap` [p.36] , readonly
A `NamedNodeMap` [p.36] containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the `Notation` [p.50] interface.
The DOM Level 1 does not support editing notations, therefore `notations` cannot be altered in any way.

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 [XML]), or is used for formal declaration of processing instruction targets (see *section 2.6* of the XML 1.0 specification [XML]). The `nodeName` attribute inherited from `Node` [p.28] is set to the declared name of the notation.

The DOM Level 1 does not support editing `Notation` nodes; they are therefore *readonly* [p.128] .

A `Notation` node does not have any parent.
IDL Definition

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

Attributes
`publicId` of type `DOMString` [p.19] , readonly
The public identifier of this notation. If the public identifier was not specified, this is `null`.

`systemId` of type `DOMString` [p.19] , readonly
The system identifier of this notation. If the system identifier was not specified, this is `null`.

Interface *Entity*

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

The `nodeName` attribute that is inherited from `Node` [p.28] contains the name of the entity.

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 `EntityReference` [p.52] nodes in the document tree.

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. When the replacement value is available, the corresponding `Entity` node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

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

The DOM Level 1 does not support editing `Entity` nodes; if a user wants to make changes to the contents of an `Entity`, every related `EntityReference` [p.52] node has to be replaced in the structure model by a clone of the `Entity`'s contents, and then the desired changes must be made to each of those clones instead. `Entity` nodes and all their descendants are *readonly* [p.128] .

An `Entity` node does not have any parent.

IDL Definition

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

Attributes

`notationName` of type `DOMString` [p.19] , readonly
: For unparsed entities, the name of the notation for the entity. For parsed entities, this is `null`.

`publicId` of type `DOMString` [p.19] , readonly
: The public identifier associated with the entity, if specified. If the public identifier was not specified, this is `null`.

`systemId` of type `DOMString` [p.19] , readonly
: The system identifier associated with the entity, if specified. If the system identifier was not specified, 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 references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. Moreover, the XML processor may completely expand references to entities while building the structure model, instead of providing `EntityReference` objects. If it does provide such objects, then for a given `EntityReference` node, it may be that there is no `Entity` [p.51] node representing the referenced entity. If such an `Entity` exists, then the child list of the `EntityReference` node is the same as that of the `Entity` node.

As for `Entity` [p.51] nodes, `EntityReference` nodes and all their descendants are *readonly* [p.128] .

The resolution of the children of the `EntityReference` (the replacement value of the referenced `Entity` [p.51]) 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 {
};
```

Interface *ProcessingInstruction*

The `ProcessingInstruction` interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.

IDL Definition

```
interface ProcessingInstruction : Node {
  readonly attribute DOMString        target;
           attribute DOMString        data;
                                        // raises(DOMException) on setting

};
```

Attributes

`data` of type `DOMString` [p.19]

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

Exceptions on setting

`DOMException` [p.20]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

`target` of type `DOMString` [p.19] , readonly

The target of this processing instruction. XML defines this as being the first token following the markup that begins the processing instruction.

2. Document Object Model HTML

Editors
Mike Champion, ArborText
Vidur Apparao, Netscape
Scott Isaacs, Microsoft (until January 1998)
Chris Wilson, Microsoft (after January 1998)
Ian Jacobs, W3C

2.1. Introduction

This section extends the Level 1 Core API to describe objects and methods specific to HTML documents [HTML4.0]. 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* [p.126] .
- 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. 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 the HTML 4.0 DTDs.

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

- An HTMLDocument interface, derived from the core `Document` [p.23] interface. `HTMLDocument` [p.55] specifies the operations and queries that can be made on a HTML document.
- An `HTMLElement` [p.59] interface, derived from the core `Element` [p.43] 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` [p.59] 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.

The interfaces found within this section are not mandatory. A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.22] interface to determine whether they are supported or not. The feature string for all the interfaces listed in this section is "HTML" and the version is "1.0".

The interfaces in this specification are designed for HTML 4.0 documents, and not for XHTML documents. Use of the HTML DOM with XHTML documents may result in incorrect processing; see Appendix C11 in the [XHTML10] for more information.

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.

2.2.1.1. 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.

2.2.1.2. 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* [p.126] 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 be *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 DOMString name);
};
```

Attributes

`length` of type `unsigned long`, readonly

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` of type `unsigned long`

The index of the node to be fetched. The index origin is 0.

Return Value

`Node` [p.28]	The `Node` at the corresponding position upon success. A value of `null` is returned if the index is out of range.

No Exceptions

`namedItem`

This method retrieves a `Node` [p.28] using a name. It first searches for a `Node` with a matching `id` attribute. If it doesn't find one, it then searches for a `Node` with a matching `name` attribute, but only on those elements that are allowed a name attribute.

Parameters

`name` of type `DOMString` [p.19]

The name of the `Node` [p.28] to be fetched.

Return Value

`Node` [p.28]	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`.

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. Besides 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 DOMString        title;
  readonly attribute DOMString        referrer;
  readonly attribute DOMString        domain;
  readonly attribute DOMString        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 DOMString        cookie;
  void               open();
  void               close();
  void               write(in DOMString text);
  void               writeln(in DOMString text);
  Element            getElementById(in DOMString elementId);
  NodeList           getElementsByName(in DOMString elementName);
};
```

Attributes

`URL` of type `DOMString` [p.19] , readonly
: The complete URI of the document.

`anchors` of type `HTMLCollection` [p.54] , readonly
: 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.

`applets` of type `HTMLCollection` [p.54] , readonly
: A collection of all the `OBJECT` elements that include applets and `APPLET` (*deprecated*) elements in a document.

`body` of type `HTMLElement` [p.59]
: The element that contains the content for the document. In documents with `BODY` contents, returns the `BODY` element. In frameset documents, this returns the outermost `FRAMESET` element.

`cookie` of type `DOMString` [p.19]
: 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`.

`domain` of type `DOMString` [p.19] , readonly
: The domain name of the server that served the document, or null if the server cannot be identified by a domain name.

`forms` of type `HTMLCollection` [p.54] , readonly
: A collection of all the forms of a document.

`images` of type `HTMLCollection` [p.54] , readonly
: A collection of all the `IMG` elements in a document. The behavior is limited to `IMG` elements for backwards compatibility.

`links` of type `HTMLCollection` [p.54] , readonly
: A collection of all `AREA` elements and anchor (`A`) elements in a document with a value for the `href` attribute.

`referrer` of type `DOMString` [p.19] , readonly
: 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).

`title` of type `DOMString` [p.19]
: The title of a document as specified by the `TITLE` element in the head of the document.

Methods

`close`
: Closes a document stream opened by `open()` and forces rendering.

No Parameters
No Return Value
No Exceptions

`getElementById`
: Returns the Element whose `id` is given by elementId. If no such element exists, returns `null`. Behavior is not defined if more than one element has this `id`.
Parameters
`elementId` of type `DOMString` [p.19]
The unique `id` value for an element.

Return Value

`Element` [p.43]	The matching element.

No Exceptions

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

`elementName` of type `DOMString` [p.19]
The `name` attribute value for an element.

Return Value

`NodeList` [p.35] The matching elements.

No Exceptions

`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.

No Parameters
No Return Value
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` of type `DOMString` [p.19]
The string to be parsed into some structure in the document structure model.

No Return Value
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` of type `DOMString` [p.19]
The string to be parsed into some structure in the document structure model.

No Return Value
No Exceptions

2.5. HTML Elements

2.5.1. Property Attributes

HTML attributes are exposed as properties on the element object. The DOM naming conventions always determine the name of the exposed property, 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* [p.126] . 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	DOMString
Value list (e.g., (left | right | center))	DOMString
one-value Value list (e.g., (disabled))	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 namespace 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 DOMString        id;
           attribute DOMString        title;
           attribute DOMString        lang;
           attribute DOMString        dir;
           attribute DOMString        className;
};
```

Attributes

`className` of type `DOMString` [p.19]
: 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.

`dir` of type `DOMString` [p.19]
: Specifies the base direction of directionally neutral text and the directionality of tables. See the dir attribute definition in HTML 4.0.

`id` of type `DOMString` [p.19]
: The element’s identifier. See the id attribute definition in HTML 4.0.

`lang` of type `DOMString` [p.19]
: Language code defined in RFC 1766. See the lang attribute definition in HTML 4.0.

`title` of type `DOMString` [p.19]
: The element’s advisory title. See the title 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 DOMString        version;
};
```

Attributes

`version` of type `DOMString` [p.19]

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 DOMString        profile;
};
```

Attributes

`profile` of type `DOMString` [p.19]

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 DOMString        charset;
           attribute DOMString        href;
           attribute DOMString        hreflang;
           attribute DOMString        media;
           attribute DOMString        rel;
           attribute DOMString        rev;
           attribute DOMString        target;
           attribute DOMString        type;
};
```

Attributes

`charset` of type `DOMString` [p.19]

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

`disabled` of type `boolean`

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

`href` of type `DOMString` [p.19]

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

`hreflang` of type `DOMString` [p.19]
Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

`media` of type `DOMString` [p.19]
Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

`rel` of type `DOMString` [p.19]
Forward link type. See the rel attribute definition in HTML 4.0.

`rev` of type `DOMString` [p.19]
Reverse link type. See the rev attribute definition in HTML 4.0.

`target` of type `DOMString` [p.19]
Frame to render the resource in. See the target attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19]
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 DOMString        text;
};
```

Attributes

`text` of type `DOMString` [p.19]
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 DOMString        content;
           attribute DOMString        httpEquiv;
           attribute DOMString        name;
           attribute DOMString        scheme;
};
```

Attributes

`content` of type `DOMString` [p.19]
Associated information. See the content attribute definition in HTML 4.0.

`httpEquiv` of type `DOMString` [p.19]
HTTP response header name. See the http-equiv attribute definition in HTML 4.0.

`name` of type `DOMString` [p.19]
Meta information name. See the name attribute definition in HTML 4.0.

`scheme` of type `DOMString` [p.19]
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 DOMString        href;
           attribute DOMString        target;
};
```

Attributes

`href` of type `DOMString` [p.19]
The base URI. See the href attribute definition in HTML 4.0.

`target` of type `DOMString` [p.19]
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 DOMString        prompt;
};
```

Attributes

`form` of type `HTMLFormElement` [p.65] , readonly
Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`prompt` of type `DOMString` [p.19]
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 DOMString        media;
           attribute DOMString        type;
};
```

Attributes

`disabled` of type `boolean`

Enables/disables the style sheet.

`media` of type `DOMString` [p.19]

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

`type` of type `DOMString` [p.19]

The content type of the style sheet language. 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 DOMString        aLink;
           attribute DOMString        background;
           attribute DOMString        bgColor;
           attribute DOMString        link;
           attribute DOMString        text;
           attribute DOMString        vLink;
};
```

Attributes

`aLink` of type `DOMString` [p.19]

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` of type `DOMString` [p.19]

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` of type `DOMString` [p.19]

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

`link` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
Document text color. See the text attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`vLink` of type `DOMString` [p.19]
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 DOMString        name;
           attribute DOMString        acceptCharset;
           attribute DOMString        action;
           attribute DOMString        enctype;
           attribute DOMString        method;
           attribute DOMString        target;
  void               submit();
  void               reset();
};
```

Attributes

`acceptCharset` of type `DOMString` [p.19]
List of character sets supported by the server. See the accept-charset attribute definition in HTML 4.0.

`action` of type `DOMString` [p.19]
Server-side form handler. See the action attribute definition in HTML 4.0.

`elements` of type `HTMLCollection` [p.54] , readonly
Returns a collection of all control elements in the form.

`enctype` of type `DOMString` [p.19]
The content type of the submitted form, generally "application/x-www-form-urlencoded". See the enctype attribute definition in HTML 4.0.

`length` of type `long`, readonly
The number of form controls in the form.

`method` of type `DOMString` [p.19]
HTTP method used to submit form. See the method attribute definition in HTML 4.0.

`name` of type `DOMString` [p.19]
Names the form.

`target` of type `DOMString` [p.19]
Frame to render the resource in. See the target attribute definition in HTML 4.0.

Methods

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

No Parameters
No Return Value
No Exceptions

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

No Parameters
No Return Value
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 DOMString        type;
           attribute long             selectedIndex;
           attribute DOMString        value;
  readonly attribute long             length;
  readonly attribute HTMLFormElement  form;
  readonly attribute HTMLCollection   options;
           attribute boolean          disabled;
           attribute boolean          multiple;
           attribute DOMString        name;
           attribute long             size;
           attribute long             tabIndex;
  void               add(in HTMLElement element,
                         in HTMLElement before)
                                        raises(DOMException);
  void               remove(in long index);
  void               blur();
  void               focus();
};
```

Attributes

`disabled` of type `boolean`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`form` of type `HTMLFormElement` [p.65] , readonly
: Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`length` of type `long`, readonly
: The number of options in this `SELECT`.

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

`name` of type `DOMString` [p.19]
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`options` of type `HTMLCollection` [p.54] , readonly
: The collection of `OPTION` elements contained by this element.

`selectedIndex` of type `long`
: The ordinal index of the selected option, starting from 0. The value -1 is returned if no element is selected. If multiple options are selected, the index of the first selected option is returned.

`size` of type `long`
: Number of visible rows. See the size attribute definition in HTML 4.0.

`tabIndex` of type `long`
: Index that represents the element’s position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19] , readonly
: The type of this form control. This is the string "select-multiple" when the multiple attribute is `true` and the string "select-one" when `false`.

`value` of type `DOMString` [p.19]
: The current form control value.

Methods

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

`element` of type `HTMLElement` [p.59]
The element to add.

`before` of type `HTMLElement`
The element to insert before, or `null` for the tail of the list.

Exceptions

`DOMException` [p.20]	NOT_FOUND_ERR: Raised if `before` is not a descendant of the `SELECT` element.

No Return Value

`blur`
Removes keyboard focus from this element.

No Parameters
No Return Value
No Exceptions

`focus`
Gives keyboard focus to this element.

No Parameters
No Return Value
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` of type `long`
The index of the item to remove, starting from 0.

No Return Value
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 DOMString        label;
};
```

Attributes

`disabled` of type `boolean`

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

`label` of type `DOMString` [p.19]

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 DOMString        text;
  readonly attribute long             index;
           attribute boolean          disabled;
           attribute DOMString        label;
           attribute boolean          selected;
           attribute DOMString        value;
};
```

Attributes

`defaultSelected` of type `boolean`

Represents the value of the HTML selected attribute. The value of this attribute does not change if the state of the corresponding form control, in an interactive user agent, changes. Changing `defaultSelected`, however, resets the state of the form control. See the selected attribute definition in HTML 4.0.

`disabled` of type `boolean`

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

`form` of type `HTMLFormElement` [p.65] , readonly

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

`index` of type `long`, readonly

The index of this `OPTION` in its parent `SELECT`, starting from 0.

`label` of type `DOMString` [p.19]

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

`selected` of type `boolean`

Represents the current state of the corresponding form control, in an interactive user agent. Changing this attribute changes the state of the form control, but does not change the value of the HTML selected attribute of the element.

`text` of type `DOMString` [p.19] , readonly
: The text contained within the option element.

`value` of type `DOMString` [p.19]
: The current form control value. See the value attribute definition in HTML 4.0.

Interface *HTMLInputElement*

Form control. *Note.* Depending upon the environment in which 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 DOMString        defaultValue;
           attribute boolean          defaultChecked;
  readonly attribute HTMLFormElement  form;
           attribute DOMString        accept;
           attribute DOMString        accessKey;
           attribute DOMString        align;
           attribute DOMString        alt;
           attribute boolean          checked;
           attribute boolean          disabled;
           attribute long             maxLength;
           attribute DOMString        name;
           attribute boolean          readOnly;
           attribute DOMString        size;
           attribute DOMString        src;
           attribute long             tabIndex;
  readonly attribute DOMString        type;
           attribute DOMString        useMap;
           attribute DOMString        value;
  void               blur();
  void               focus();
  void               select();
  void               click();
};
```

Attributes

`accept` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`align` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

`checked` of type `boolean`
: When the `type` attribute of the element has the value "Radio" or "Checkbox", this represents the current state of the form control, in an interactive user agent. Changes to this attribute change the state of the form control, but do not change the value of the HTML value attribute of the element.

`defaultChecked` of type `boolean`
: When `type` has the value "Radio" or "Checkbox", this represents the HTML checked attribute of the element. The value of this attribute does not change if the state of the corresponding form control, in an interactive user agent, changes. Changes to this attribute, however, resets the state of the form control. See the checked attribute definition in HTML 4.0.

`defaultValue` of type `DOMString` [p.19]
: When the `type` attribute of the element has the value "Text", "File" or "Password", this represents the HTML value attribute of the element. The value of this attribute does not change if the contents of the corresponding form control, in an interactive user agent, changes. Changing this attribute, however, resets the contents of the form control. See the value attribute definition in HTML 4.0.

`disabled` of type `boolean`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`form` of type `HTMLFormElement` [p.65] , readonly
: Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`maxLength` of type `long`
: 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` of type `DOMString` [p.19]
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`readOnly` of type `boolean`
: This control is read-only. Relevant only when `type` has the value "Text" or "Password". See the readonly attribute definition in HTML 4.0.

`size` of type `DOMString` [p.19]
: Size information. The precise meaning is specific to each type of field. See the size attribute definition in HTML 4.0.

`src` of type `DOMString` [p.19]
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` of type `long`
Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19] , readonly
The type of control created. See the type attribute definition in HTML 4.0.

`useMap` of type `DOMString` [p.19]
Use client-side image map. See the usemap attribute definition in HTML 4.0.

`value` of type `DOMString` [p.19]
When the `type` attribute of the element has the value "Text", "File" or "Password", this represents the current contents of the corresponding form control, in an interactive user agent. Changing this attribute changes the contents of the form control, but does not change the value of the HTML value attribute of the element. When the `type` attribute of the element has the value "Button", "Hidden", "Submit", "Reset", "Image", "Checkbox" or "Radio", this represents the HTML value attribute of the element. See the value attribute definition in HTML 4.0.

Methods

`blur`
Removes keyboard focus from this element.

No Parameters
No Return Value
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".

No Parameters
No Return Value
No Exceptions

`focus`
Gives keyboard focus to this element.

No Parameters
No Return Value
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".

No Parameters
No Return Value
No Exceptions

Interface *HTMLTextAreaElement*

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

IDL Definition

```
interface HTMLTextAreaElement : HTMLElement {
           attribute DOMString        defaultValue;
  readonly attribute HTMLFormElement  form;
           attribute DOMString        accessKey;
           attribute long             cols;
           attribute boolean          disabled;
           attribute DOMString        name;
           attribute boolean          readOnly;
           attribute long             rows;
           attribute long             tabIndex;
  readonly attribute DOMString        type;
           attribute DOMString        value;
  void               blur();
  void               focus();
  void               select();
};
```

Attributes

`accessKey` of type `DOMString` [p.19]
A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

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

`defaultValue` of type `DOMString` [p.19]
Represents the contents of the element. The value of this attribute does not change if the contents of the corresponding form control, in an interactive user agent, changes. Changing this attribute, however, resets the contents of the form control.

`disabled` of type `boolean`
The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`form` of type `HTMLFormElement` [p.65] , readonly
Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`name` of type `DOMString` [p.19]
Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`readOnly` of type `boolean`
This control is read-only. See the readonly attribute definition in HTML 4.0.

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

`tabIndex` of type `long`
Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19] , readonly
The type of this form control. This the string "textarea".

`value` of type `DOMString` [p.19]
Represents the current contents of the corresponding form control, in an interactive user agent. Changing this attribute changes the contents of the form control, but does not change the contents of the element. If the entirety of the data can not fit into a single `DOMString` [p.19] , the implementation may truncate the data.

Methods

`blur`
Removes keyboard focus from this element.

No Parameters
No Return Value
No Exceptions

`focus`
Gives keyboard focus to this element.

No Parameters
No Return Value
No Exceptions

`select`
Select the contents of the TEXTAREA.

No Parameters
No Return Value
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 DOMString        accessKey;
           attribute boolean          disabled;
           attribute DOMString        name;
           attribute long             tabIndex;
  readonly attribute DOMString        type;
           attribute DOMString        value;
};
```

Attributes

`accessKey` of type `DOMString` [p.19]
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`disabled` of type `boolean`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`form` of type `HTMLFormElement` [p.65] , readonly
: Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`name` of type `DOMString` [p.19]
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`tabIndex` of type `long`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19] , readonly
: The type of button. See the type attribute definition in HTML 4.0.

`value` of type `DOMString` [p.19]
: 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 DOMString        accessKey;
           attribute DOMString        htmlFor;
};
```

Attributes

`accessKey` of type `DOMString` [p.19]
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`form` of type `HTMLFormElement` [p.65] , readonly
: Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`htmlFor` of type `DOMString` [p.19]
: 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` of type `HTMLFormElement` [p.65] , readonly
: 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 DOMString        accessKey;
           attribute DOMString        align;
};
```

Attributes

`accessKey` of type `DOMString` [p.19]
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`align` of type `DOMString` [p.19]
: Text alignment relative to `FIELDSET`. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`form` of type `HTMLFormElement` [p.65] , readonly
: Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

Interface *HTMLUListElement*

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

IDL Definition

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

Attributes

`compact` of type `boolean`

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

`type` of type `DOMString` [p.19]

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 DOMString        type;
};
```

Attributes

`compact` of type `boolean`

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

`start` of type `long`

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

`type` of type `DOMString` [p.19]

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` of type `boolean`

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` of type `boolean`

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` of type `boolean`

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 DOMString        type;
           attribute long             value;
};
```

Attributes

`type` of type `DOMString` [p.19]

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

`value` of type `long`
: 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 *HTMLDivElement*

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

IDL Definition

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

Attributes

`align` of type `DOMString` [p.19]
: 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 DOMString        align;
};
```

Attributes

`align` of type `DOMString` [p.19]
: 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 DOMString        align;
};
```

Attributes

`align` of type `DOMString` [p.19]
: 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 DOMString        cite;
};
```

Attributes

`cite` of type `DOMString` [p.19]

A URI designating 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` of type `long`

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 DOMString        clear;
};
```

Attributes

`clear` of type `DOMString` [p.19]

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 DOMString        color;
           attribute DOMString        face;
           attribute DOMString        size;
};
```

Attributes

`color` of type `DOMString` [p.19]
: Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`face` of type `DOMString` [p.19]
: Font face identifier. See the face attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`size` of type `DOMString` [p.19]
: 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 DOMString        color;
           attribute DOMString        face;
           attribute DOMString        size;
};
```

Attributes

`color` of type `DOMString` [p.19]
: Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`face` of type `DOMString` [p.19]
: Font face identifier. See the face attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`size` of type `DOMString` [p.19]
: 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 DOMString        align;
           attribute boolean          noShade;
           attribute DOMString        size;
           attribute DOMString        width;
};
```

Attributes

`align` of type `DOMString` [p.19]
: Align the rule on the page. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`noShade` of type `boolean`
: 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` of type `DOMString` [p.19]
: The height of the rule. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`width` of type `DOMString` [p.19]
: 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 DOMString        cite;
           attribute DOMString        dateTime;
};
```

Attributes

`cite` of type `DOMString` [p.19]
: A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

`dateTime` of type `DOMString` [p.19]
: 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 DOMString        accessKey;
           attribute DOMString        charset;
           attribute DOMString        coords;
           attribute DOMString        href;
           attribute DOMString        hreflang;
           attribute DOMString        name;
           attribute DOMString        rel;
           attribute DOMString        rev;
```

```
           attribute DOMString        shape;
           attribute long             tabIndex;
           attribute DOMString        target;
           attribute DOMString        type;
  void               blur();
  void               focus();
};
```

Attributes

`accessKey` of type `DOMString` [p.19]
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`charset` of type `DOMString` [p.19]
: The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

`coords` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: The URI of the linked resource. See the href attribute definition in HTML 4.0.

`hreflang` of type `DOMString` [p.19]
: Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

`name` of type `DOMString` [p.19]
: Anchor name. See the name attribute definition in HTML 4.0.

`rel` of type `DOMString` [p.19]
: Forward link type. See the rel attribute definition in HTML 4.0.

`rev` of type `DOMString` [p.19]
: Reverse link type. See the rev attribute definition in HTML 4.0.

`shape` of type `DOMString` [p.19]
: The shape of the active area. The coordinates are given by `coords`. See the shape attribute definition in HTML 4.0.

`tabIndex` of type `long`
: Index that represents the element’s position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`target` of type `DOMString` [p.19]
: Frame to render the resource in. See the target attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19]
: Advisory content type. See the type attribute definition in HTML 4.0.

Methods

`blur`

Removes keyboard focus from this element.

No Parameters
No Return Value
No Exceptions

`focus`

Gives keyboard focus to this element.

No Parameters
No Return Value
No Exceptions

Interface *HTMLImageElement*

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

IDL Definition

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

Attributes

`align` of type `DOMString` [p.19]

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` of type `DOMString` [p.19]

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

`border` of type `DOMString` [p.19]

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

`height` of type `DOMString` [p.19]
Override height. See the height attribute definition in HTML 4.0.

`hspace` of type `DOMString` [p.19]
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` of type `boolean`
Use server-side image map. See the ismap attribute definition in HTML 4.0.

`longDesc` of type `DOMString` [p.19]
URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`lowSrc` of type `DOMString` [p.19]
URI designating the source of this image, for low-resolution output.

`name` of type `DOMString` [p.19]
The name of the element (for backwards compatibility).

`src` of type `DOMString` [p.19]
URI designating the source of this image. See the src attribute definition in HTML 4.0.

`useMap` of type `DOMString` [p.19]
Use client-side image map. See the usemap attribute definition in HTML 4.0.

`vspace` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
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 DOMString        code;
           attribute DOMString        align;
           attribute DOMString        archive;
           attribute DOMString        border;
           attribute DOMString        codeBase;
           attribute DOMString        codeType;
           attribute DOMString        data;
           attribute boolean          declare;
```

```
            attribute DOMString        height;
            attribute DOMString        hspace;
            attribute DOMString        name;
            attribute DOMString        standby;
            attribute long             tabIndex;
            attribute DOMString        type;
            attribute DOMString        useMap;
            attribute DOMString        vspace;
            attribute DOMString        width;
};
```

Attributes

`align` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: Space-separated list of archives. See the archive attribute definition in HTML 4.0.

`border` of type `DOMString` [p.19]
: Width of border around the object. See the border attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`code` of type `DOMString` [p.19]
: Applet class file. See the `code` attribute for HTMLAppletElement.

`codeBase` of type `DOMString` [p.19]
: Base URI for `classid`, `data`, and `archive` attributes. See the codebase attribute definition in HTML 4.0.

`codeType` of type `DOMString` [p.19]
: Content type for data downloaded via `classid` attribute. See the codetype attribute definition in HTML 4.0.

`data` of type `DOMString` [p.19]
: A URI specifying the location of the object's data. See the data attribute definition in HTML 4.0.

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

`form` of type `HTMLFormElement` [p.65] , readonly
: Returns the `FORM` element containing this control. Returns `null` if this control is not within the context of a form.

`height` of type `DOMString` [p.19]
: Override height. See the height attribute definition in HTML 4.0.

`hspace` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`standby` of type `DOMString` [p.19]
: Message to render while loading the object. See the standby attribute definition in HTML 4.0.

`tabIndex` of type `long`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19]
: Content type for data downloaded via `data` attribute. See the type attribute definition in HTML 4.0.

`useMap` of type `DOMString` [p.19]
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`vspace` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: 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 DOMString        name;
           attribute DOMString        type;
           attribute DOMString        value;
           attribute DOMString        valueType;
};
```

Attributes

`name` of type `DOMString` [p.19]
: The name of a run-time parameter. See the name attribute definition in HTML 4.0.

`type` of type `DOMString` [p.19]
: Content type for the `value` attribute when `valuetype` has the value "ref". See the type attribute definition in HTML 4.0.

`value` of type `DOMString` [p.19]
The value of a run-time parameter. See the value attribute definition in HTML 4.0.

`valueType` of type `DOMString` [p.19]
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 DOMString        align;
           attribute DOMString        alt;
           attribute DOMString        archive;
           attribute DOMString        code;
           attribute DOMString        codeBase;
           attribute DOMString        height;
           attribute DOMString        hspace;
           attribute DOMString        name;
           attribute DOMString        object;
           attribute DOMString        vspace;
           attribute DOMString        width;
};
```

Attributes

`align` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
Comma-separated archive list. See the archive attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`code` of type `DOMString` [p.19]
Applet class file. See the code attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`codeBase` of type `DOMString` [p.19]
Optional base URI for applet. See the codebase attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`height` of type `DOMString` [p.19]
Override height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`hspace` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
The name of the applet. See the name attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`object` of type `DOMString` [p.19]
Serialized applet file. See the object attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`vspace` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
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 DOMString        name;
};
```

Attributes

`areas` of type `HTMLCollection` [p.54] , readonly
The list of areas defined for the image map.

`name` of type `DOMString` [p.19]
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 DOMString        accessKey;
           attribute DOMString        alt;
           attribute DOMString        coords;
           attribute DOMString        href;
           attribute boolean          noHref;
           attribute DOMString        shape;
           attribute long             tabIndex;
           attribute DOMString        target;
};
```

Attributes

`accessKey` of type `DOMString` [p.19]
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`alt` of type `DOMString` [p.19]
: Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

`coords` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: The URI of the linked resource. See the href attribute definition in HTML 4.0.

`noHref` of type `boolean`
: Specifies that this area is inactive, i.e., has no associated action. See the nohref attribute definition in HTML 4.0.

`shape` of type `DOMString` [p.19]
: The shape of the active area. The coordinates are given by `coords`. See the shape attribute definition in HTML 4.0.

`tabIndex` of type `long`
: Index that represents the element’s position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`target` of type `DOMString` [p.19]
: 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 DOMString        text;
           attribute DOMString        htmlFor;
           attribute DOMString        event;
           attribute DOMString        charset;
           attribute boolean          defer;
           attribute DOMString        src;
           attribute DOMString        type;
};
```

Attributes

`charset` of type `DOMString` [p.19]
: The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

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

`event` of type `DOMString` [p.19]
: *Reserved for future use.*

`htmlFor` of type `DOMString` [p.19]
: *Reserved for future use.*

`src` of type `DOMString` [p.19]
: URI designating an external script. See the src attribute definition in HTML 4.0.

`text` of type `DOMString` [p.19]
: The script content of the element.

`type` of type `DOMString` [p.19]
: 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;
  readonly attribute HTMLCollection   tBodies;
           attribute DOMString        align;
           attribute DOMString        bgColor;
           attribute DOMString        border;
           attribute DOMString        cellPadding;
```

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

Attributes

`align` of type `DOMString` [p.19]
: 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` of type `DOMString` [p.19]
: Cell background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`border` of type `DOMString` [p.19]
: The width of the border around the table. See the border attribute definition in HTML 4.0.

`caption` of type `HTMLTableCaptionElement` [p.95]
: Returns the table's `CAPTION`, or void if none exists.

`cellPadding` of type `DOMString` [p.19]
: Specifies the horizontal and vertical space between cell content and cell borders. See the cellpadding attribute definition in HTML 4.0.

`cellSpacing` of type `DOMString` [p.19]
: Specifies the horizontal and vertical separation between cells. See the cellspacing attribute definition in HTML 4.0.

`frame` of type `DOMString` [p.19]
: Specifies which external table borders to render. See the frame attribute definition in HTML 4.0.

`rows` of type `HTMLCollection` [p.54] , readonly
: Returns a collection of all the rows in the table, including all in `THEAD`, `TFOOT`, all `TBODY` elements.

`rules` of type `DOMString` [p.19]
: Specifies which internal table borders to render. See the rules attribute definition in HTML 4.0.

`summary` of type `DOMString` [p.19]
Description about the purpose or structure of a table. See the summary attribute definition in HTML 4.0.

`tBodies` of type `HTMLCollection` [p.54] , readonly
Returns a collection of the defined table bodies.

`tFoot` of type `HTMLTableSectionElement` [p.96]
Returns the table's `TFOOT`, or `null` if none exists.

`tHead` of type `HTMLTableSectionElement` [p.96]
Returns the table's `THEAD`, or `null` if none exists.

`width` of type `DOMString` [p.19]
Specifies the desired table width. See the width attribute definition in HTML 4.0.

Methods

`createCaption`
Create a new table caption object or return an existing one.
Return Value

`HTMLElement` [p.59]	A `CAPTION` element.

No Parameters
No Exceptions

`createTFoot`
Create a table footer row or return an existing one.
Return Value

`HTMLElement` [p.59]	A footer element (`TFOOT`).

No Parameters
No Exceptions

`createTHead`
Create a table header row or return an existing one.
Return Value

`HTMLElement` [p.59]	A new table header element (`THEAD`).

No Parameters
No Exceptions

`deleteCaption`
Delete the table caption, if one exists.

No Parameters
No Return Value
No Exceptions

`deleteRow`
Delete a table row.
Parameters
`index` of type `long`
The index of the row to be deleted. This index starts from 0 and is relative to all the rows contained inside the table, regardless of section parentage.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified index is greater than or equal to the number of rows or if the index is negative.

No Return Value

`deleteTFoot`
Delete the footer from the table, if one exists.

No Parameters
No Return Value
No Exceptions

`deleteTHead`
Delete the header from the table, if one exists.

No Parameters
No Return Value
No Exceptions

`insertRow`
Insert a new empty row in the table. The new row is inserted immediately before and in the same section as the current `index`th row in the table. If `index` is equal to the number of rows, the new row is appended. In addition, when the table is empty the row is inserted into a `TBODY` which is created and inserted into the table. *Note.* A table row cannot be empty according to HTML 4.0 Recommendation.
Parameters
`index` of type `long`
The row number where to insert a new row. This index starts from 0 and is relative to all the rows contained inside the table, regardless of section parentage.

Return Value

`HTMLElement` [p.59]	The newly created row.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified index is greater than the number of rows or if the index is negative.

Interface *HTMLTableCaptionElement*

Table caption See the CAPTION element definition in HTML 4.0.

IDL Definition

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

Attributes

`align` of type `DOMString` [p.19]

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 DOMString        align;
           attribute DOMString        ch;
           attribute DOMString        chOff;
           attribute long             span;
           attribute DOMString        vAlign;
           attribute DOMString        width;
};
```

Attributes

`align` of type `DOMString` [p.19]

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

`ch` of type `DOMString` [p.19]

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

`chOff` of type `DOMString` [p.19]

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

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

`vAlign` of type `DOMString` [p.19]
: Vertical alignment of cell data in column. See the valign attribute definition in HTML 4.0.

`width` of type `DOMString` [p.19]
: 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 DOMString        align;
           attribute DOMString        ch;
           attribute DOMString        chOff;
           attribute DOMString        vAlign;
  readonly attribute HTMLCollection   rows;
  HTMLElement        insertRow(in long index)
                                        raises(DOMException);
  void               deleteRow(in long index)
                                        raises(DOMException);
};
```

Attributes

`align` of type `DOMString` [p.19]
: Horizontal alignment of data in cells. See the `align` attribute for HTMLTheadElement for details.

`ch` of type `DOMString` [p.19]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.19]
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`rows` of type `HTMLCollection` [p.54] , readonly
: The collection of rows in this table section.

`vAlign` of type `DOMString` [p.19]
: Vertical alignment of data in cells. See the `valign` attribute for HTMLTheadElement for details.

Methods

`deleteRow`
: Delete a row from this section.

  Parameters

`index` of type `long`
The index of the row to be deleted. This index starts from 0 and is relative only to the rows contained inside this section, not all the rows in the table.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified index is greater than or equal to the number of rows or if the index is negative.

No Return Value

`insertRow`
Insert a row into this section. The new row is inserted immediately before the current `index`th row in this section. If `index` is equal to the number of rows in this section, the new row is appended.
Parameters
`index` of type `long`
The row number where to insert a new row. This index starts from 0 and is relative only to the rows contained inside this section, not all the rows in the table.

Return Value

`HTMLElement` [p.59]	The newly created row.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified index is greater than the number of rows or if the index is negative.

Interface *HTMLTableRowElement*

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

```
interface HTMLTableRowElement : HTMLElement {
  readonly attribute long            rowIndex;
  readonly attribute long            sectionRowIndex;
  readonly attribute HTMLCollection  cells;
           attribute DOMString       align;
           attribute DOMString       bgColor;
           attribute DOMString       ch;
           attribute DOMString       chOff;
           attribute DOMString       vAlign;
  HTMLElement       insertCell(in long index)
```

```
                                        raises(DOMException);
  void               deleteCell(in long index)
                                        raises(DOMException);
};
```

Attributes

`align` of type `DOMString` [p.19]
: Horizontal alignment of data within cells of this row. See the align attribute definition in HTML 4.0.

`bgColor` of type `DOMString` [p.19]
: Background color for rows. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`cells` of type `HTMLCollection` [p.54] , readonly
: The collection of cells in this row.

`ch` of type `DOMString` [p.19]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.19]
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`rowIndex` of type `long`, readonly
: The index of this row, relative to the entire table, starting from 0. This is in document tree order and not display order. The `rowIndex` does not take into account sections (`THEAD`, `TFOOT`, or `TBODY`) within the table.

`sectionRowIndex` of type `long`, readonly
: The index of this row, relative to the current section (`THEAD`, `TFOOT`, or `TBODY`), starting from 0.

`vAlign` of type `DOMString` [p.19]
: Vertical alignment of data within cells of this row. See the valign attribute definition in HTML 4.0.

Methods

`deleteCell`
: Delete a cell from the current row.

Parameters

`index` of type `long`
: The index of the cell to delete, starting from 0.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified `index` is greater than or equal to the number of cells or if the index is negative.

No Return Value

`insertCell`

Insert an empty `TD` cell into this row. If `index` is equal to the number of cells, the new cell is appended.

Parameters

`index` of type `long`

The place to insert the cell, starting from 0.

Return Value

`HTMLElement` [p.59]	The newly created cell.

Exceptions

`DOMException` [p.20]	INDEX_SIZE_ERR: Raised if the specified `index` is greater than the number of cells or if the index is negative.

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 {
  readonly attribute long             cellIndex;
           attribute DOMString        abbr;
           attribute DOMString        align;
           attribute DOMString        axis;
           attribute DOMString        bgColor;
           attribute DOMString        ch;
           attribute DOMString        chOff;
           attribute long             colSpan;
           attribute DOMString        headers;
           attribute DOMString        height;
           attribute boolean          noWrap;
           attribute long             rowSpan;
           attribute DOMString        scope;
           attribute DOMString        vAlign;
           attribute DOMString        width;
};
```

Attributes

`abbr` of type `DOMString` [p.19]

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

`align` of type `DOMString` [p.19]

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

`axis` of type `DOMString` [p.19]
: Names group of related headers. See the axis attribute definition in HTML 4.0.

`bgColor` of type `DOMString` [p.19]
: Cell background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`cellIndex` of type `long`, readonly
: The index of this cell in the row, starting from 0. This index is in document tree order and not display order.

`ch` of type `DOMString` [p.19]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.19]
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`colSpan` of type `long`
: Number of columns spanned by cell. See the colspan attribute definition in HTML 4.0.

`headers` of type `DOMString` [p.19]
: List of `id` attribute values for header cells. See the headers attribute definition in HTML 4.0.

`height` of type `DOMString` [p.19]
: Cell height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`noWrap` of type `boolean`
: Suppress word wrapping. See the nowrap attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`rowSpan` of type `long`
: Number of rows spanned by cell. See the rowspan attribute definition in HTML 4.0.

`scope` of type `DOMString` [p.19]
: Scope covered by header cells. See the scope attribute definition in HTML 4.0.

`vAlign` of type `DOMString` [p.19]
: Vertical alignment of data in cell. See the valign attribute definition in HTML 4.0.

`width` of type `DOMString` [p.19]
: 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 DOMString        cols;
           attribute DOMString        rows;
};
```

Attributes

`cols` of type `DOMString` [p.19]

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

`rows` of type `DOMString` [p.19]

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 DOMString        frameBorder;
           attribute DOMString        longDesc;
           attribute DOMString        marginHeight;
           attribute DOMString        marginWidth;
           attribute DOMString        name;
           attribute boolean          noResize;
           attribute DOMString        scrolling;
           attribute DOMString        src;
};
```

Attributes

`frameBorder` of type `DOMString` [p.19]

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

`longDesc` of type `DOMString` [p.19]

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

`marginHeight` of type `DOMString` [p.19]

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

`marginWidth` of type `DOMString` [p.19]

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

`name` of type `DOMString` [p.19]

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

`noResize` of type `boolean`
When true, forbid user from resizing frame. See the noresize attribute definition in HTML 4.0.

`scrolling` of type `DOMString` [p.19]
Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

`src` of type `DOMString` [p.19]
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 DOMString        align;
           attribute DOMString        frameBorder;
           attribute DOMString        height;
           attribute DOMString        longDesc;
           attribute DOMString        marginHeight;
           attribute DOMString        marginWidth;
           attribute DOMString        name;
           attribute DOMString        scrolling;
           attribute DOMString        src;
           attribute DOMString        width;
};
```

Attributes

`align` of type `DOMString` [p.19]
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` of type `DOMString` [p.19]
Request frame borders. See the frameborder attribute definition in HTML 4.0.

`height` of type `DOMString` [p.19]
Frame height. See the height attribute definition in HTML 4.0.

`longDesc` of type `DOMString` [p.19]
URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`marginHeight` of type `DOMString` [p.19]
Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

`marginWidth` of type `DOMString` [p.19]
Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

`name` of type `DOMString` [p.19]
: The frame name (object of the `target` attribute). See the name attribute definition in HTML 4.0.

`scrolling` of type `DOMString` [p.19]
: Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

`src` of type `DOMString` [p.19]
: A URI designating the initial frame contents. See the src attribute definition in HTML 4.0.

`width` of type `DOMString` [p.19]
: Frame width. See the width attribute definition in HTML 4.0.

Appendix A: Changes

Editors
Philippe Le Hégaret, W3C

This appendix contains the changes from the 1 October 1998 specification.

A.1: Changes in the "What is the Document Object Model?"

Introduction [p.11]

The link to the CORBA 2.2 specification was broken.
Also, the first item of the list about "DOM Interfaces and DOM Implementations" had text about "read-only functions" which was changed to "read-only attributes".

What the Document Object Model is [p.11]

The following sentence was missing:

> Each document contains zero or one doctype nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document.

The following statement has been modified:

> 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.

It now reads:

> 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, in accordance with the XML Information Set [Infoset].

with the following note:

Note: There may be some variations depending on the parser being used to build the DOM. For instance, the DOM may not contain whitespaces in element content if the parser discards them.

The compliance paragraph has been moved into a specific section (see Compliance [p.14]).

What the Document Object Model is not [p.13]

The following statement has been modified:

> 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.

It now reads:

The Document Object Model does not define what information in a document is relevant or how information in a document is structured. For XML, this is specified by the W3C XML Information Set [Infoset]. The DOM is simply an API to this information set.

A.2: Changes in the Document Object Model Core

Section 1.1.5. The DOMString type [p.19]

The `DOMString` [p.19] type was defined as "a sequence of 16-bit quantities". Instead, it is now defined as "sequence of 16-bit units".
Then, the description contained the following sentences:

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 `DOMString` [p.19] (a high surrogate and a low surrogate).

This now reads:

Please note that for both HTML and XML, the document character set (and therefore the notation of numeric character references) is based on UCS [ISO-10646]. A single numeric character reference in a source document may therefore in some cases correspond to two 16-bit units in a `DOMString` [p.19] (a high surrogate and a low surrogate).

Section 1.1.6. String comparisons in the DOM [p.20]

The title of this section, which was *Case sensitivity in the DOM*, was changed to be more accurate.
The first paragraph ended with the following:

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 DOMString. As such, the DOM assumes that any normalizations will take place in the processor, before the DOM structures are built.

This sentence now reads:

For the purposes of the DOM, string matching is performed purely by binary comparison of the *16-bit units* [p.125] of the DOMString. In addition, the DOM assumes that any normalizations take place in the processor, before the DOM structures are built.

With the following note:

Note: Besides case folding, there are additional normalizations that can be applied to text. The W3C I18N Working Group is in the process of defining exactly which normalizations are necessary, and where they should be applied. The W3C I18N Working Group expects to require early normalization, which means that data read into the DOM is assumed to already be normalized. The DOM and applications built on top of it in this case only have to assure that text remains normalized when

being changed. For further details, please see [Charmod].

Section 1.2 Interface Attr [p.42]

The fact that setting the `value` attribute raises a `NO_MODIFICATION_ALLOWED_ERR` DOMException [p.20] when the node is readonly was added.

A sentence has been added at the end of the first paragraph of the description of the `value` attribute:

> See also the method `getAttribute` on the `Element` [p.43] interface.

also at the end of the second paragraph:

> I.e. any characters that an XML processor would recognize as markup are instead treated as literal text. See also the method `setAttribute` on the `Element` [p.43] interface.

And on the `specified` attribute:

> If the attribute is not associated to any element (i.e. because it was just created or was obtained from some removal or cloning operation) `specified` is `true`.

Section 1.2 Interface CharacterData [p.38]

The following paragraph has been added:

> As explained in the `DOMString` [p.19] interface, text strings in the DOM are represented in UTF-16, i.e. as a sequence of 16-bit units. In the following, the term *16-bit units* [p.125] is used whenever necessary to indicate that indexing on CharacterData is done in 16-bit units.

The description of the `length` attribute read:

> The number of characters that are available through data and the `substringData` method below.

It now reads:

> The number of *16-bit units* [p.125] that are available through data and the `substringData` method below.

The description of the `count` parameter of the `substringData` method read:

> The number of characters to extract.

It now reads:

> The number of 16-bit units to extract.

Then the description of the return value read:

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

It now reads:

> The specified substring. If the sum of `offset` and `count` exceeds the `length`, then all 16-bit units to the end of the data are returned.

The exception INDEX_SIZE_ERR was said to be:

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

Instead this now reads:

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

The description of the `insertData` method read:

> Insert a string at the specified character offset.

Instead it now reads:

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

Then the description of the `offset` parameter read:

> The character offset at which to insert.

when it now reads:

> The 16-bit unit offset at which to insert.

The exception INDEX_SIZE_ERR was said to be:

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

Instead this now reads:

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

The description of the `deleteData` method read:

> Remove a range of characters from the node.

Instead it now reads:

> Remove a range of *16-bit units* [p.125] from the node.

Then the description of the `count` parameter read:

> 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.

when it now reads:

> The number of 16-bit units to delete. If the sum of `offset` and `count` exceeds `length` then all 16-bit units from `offset` to the end of the data are deleted.

The description of the `replaceData` method read:

> Replace the characters starting at the specified character offset with the specified string.

instead it now reads:

> Replace the characters starting at the specified *16-bit unit* [p.125] offset with the specified string.

Then the description of the `count` parameter read:

> 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

when it now reads:

> The number of 16-bit units to replace. If the sum of `offset` and `count` exceeds `length`, then all 16-bit units to the end of the data are replaced

The exception INDEX_SIZE_ERR was said to be:

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

Instead this now reads:

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

Section 1.2 Interface DOMImplementation [p.22]

The definition of the `feature` parameter read:

> The package name of the feature to test. In Level 1, the legal values are "HTML" and "XML" (case-insensitive).

This now reads:

> The name of the feature to test (case-insensitive). The values used by DOM features are defined throughout this specification and listed i n the Compliance [p.14] section. The name must be an *XML name* [p.128] . To avoid possible conflicts, as a convention, names referring to features defined outside the DOM specification should be made unique by reversing the name of the Internet domain name of the person (or the organization that the person belongs to) who defines the feature, component by component, and using this as a prefix. For instance, the W3C SYMM Working Group defines the feature "org.w3c.dom.smil".

The definition of the `version` parameter read:

> This is the version number of the package name to test. In Level 1, this is the string "1.0".

This now reads:

> This is the version number of the feature to test. In Level 1, this is the string "1.0".

Section 1.2 Interface Document [p.23]

The description of the `createElement` method was missing the following piece:

> In addition, if there are known attributes with default values, Attr nodes representing them are automatically created and attached to the element.

The description of the `createEntityReference` method was missing the following piece:

> In addition, if the referenced entity is known, the child list of the `EntityReference` [p.52] node is made the same as that of the corresponding `Entity` [p.51] node.

The description of the `doctype` attribute was missing the following piece:

> The DOM Level 1 does not support editing the Document Type Declaration, therefore `docType` cannot be altered in any way, including through the use of methods, such as `insertNode` or `removeNode`, which are inherited from the `Node` [p.28] interface.

The description of the `createAttribute` method was said to be:

> Creates an `Attr` [p.42] of the given name. Note that the `Attr` instance can then be set on an `Element` [p.43] using the `setAttribute` method.

it is now:

> Creates an `Attr` [p.42] of the given name. Note that the `Attr` instance can then be set on an `Element` [p.43] using the `setAttributeNode` method.

The description of the return value was missing:

> The value of the attribute is the emtpy string.

The exception INVALID_CHARACTER_ERR for `createElement`, `createAttribute`, `createEntityReference` and `createProcessingInstruction` methods was said to be:

> Raised if the specified name contains an invalid character.

Instead this now reads:

> Raised if the specified name contains an illegal character.

Section 1.2 Interface DocumentType [p.49]

The description of the `entities` attribute has been modified:

> A `NamedNodeMap` [p.36] containing the general entities, both external and internal, declared in the DTD. Duplicates are discarded.

It now reads:

> A `NamedNodeMap` [p.36] containing the general entities, both external and internal, declared in the DTD. Parameter entities are not contained. Duplicates are discarded.

The example has been modified as follows:

```
<!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`. [...]

It is now:

```
<!DOCTYPE ex SYSTEM "ex.dtd" [
  <!ENTITY foo "foo">
  <!ENTITY bar "bar">
  <!ENTITY bar "bar2">
  <!ENTITY % baz "baz">
]>
<ex/>
```

> the interface provides access to `foo` and the first declaration of `bar` but not the second declaration of `bar` or `baz`. [...]

Section 1.2 Interface Element [p.43]

The following example has been removed:

> By far the vast majority of objects (apart from text) that authors encounter when traversing a document are `Element` [p.43] nodes. Assume the following XML document:

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

When represented using DOM, the top node is a `Document` [p.23] node containing an `Element` [p.43] node for "elementExample" which contains two child `Element` nodes, one for "subelement1" and one for "subelement2". "subelement1" contains no child nodes.

The following sentence has been added:

The `Element` [p.43] interface represents an element in an HTML or XML document.

The description read:

the generic Node interface method `getAttributes` may be used to retrieve the set of all attributes for an element.

However, there is no `getAttributes` method per se, although it may exist in some language binding such as the java one. So this section now reads:

the generic Node interface attribute `attributes` may be used to retrieve the set of all attributes for an element.

The `removeAttribute` method description read:

If the removed attribute has a default value it is immediately replaced.

This now reads:

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

The `removeAttributeNode` method description read:

Removes the specified attribute.

This now reads:

Removes the specified attribute. If the removed `Attr` [p.42] has a default value it is immediately replaced.

The description of the `oldAttr` has been chagned according to the previous change:

The `Attr` [p.42] node to remove from the attribute list. If the removed `Attr` has a default value it is immediately replaced.

This now reads:

> The `Attr` [p.42] node to remove from the attribute list.

In addition, the following note was added to the description of the `normalize` method:

> **Note:** In cases where the document contains `CDATASections` [p.48] , the normalize operation alone may not be sufficient, since XPointers do not differentiate between `Text` [p.47] nodes and `CDATASection` [p.48] nodes.

And the following change was made to the description of the same method:

> Puts all `Text` [p.47] nodes in the full depth of the sub-tree underneath this `Element` [p.43] 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 now reads:

> Puts all `Text` [p.47] nodes in the full depth of the sub-tree underneath this `Element` [p.43] , including attribute nodes, 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.

The exception INVALID_CHARACTER_ERR for the `setAttribute` method was said to be:

> Raised if the specified name contains an invalid character.

Instead this now reads:

> Raised if the specified name contains an illegal character.

The description for `setAttributeNode` was said to be:

> Adds a new attribute.

Instead this now reads:

> Adds a new attribute node.

The description for `setAttributeNode` return value was said to be:

> If the `newAttr` attribute replaces an existing attribute with the same name, the previously existing `Attr` [p.42] node is returned, otherwise `null` is returned.

Instead this now reads:

> If the `newAttr` attribute replaces an existing attribute, the replaced `Attr` [p.42] node is returned, otherwise `null` is returned.

Section 1.2 Interface DOMException [p.20]

The following note has been added for the `ExceptionCode` group:

> **Note:** Other numeric codes are reserved for W3C for possible future use.

The description of the INVALID_CHARACTER_ERR read:

> If an invalid character is specified, such as in a name.

Instead it now reads:

> If an invalid or illegal character is specified, such as in a name. See *production 2* in the XML specification for the definition of a legal character, and *production 5* for the definition of a legal name character.

Section 1.2 Interface NamedNodeMap [p.36]

The description of the `setNamedItem` was missing (moved from the description of the `arg` parameter):

> If a node with that name is already present in this map, it is replaced by the new one.

The description of the return value of the `removeNamedItem` method read:

> The node removed from the map or null if no node with such a name exists.

But this error case is already handled by having the method to throw a NOT_FOUND_ERR DOMException. So this section now simply reads:

> The node removed from the map if a node with such a name exists.

The description of the `removeNamedItem` method now includes the following note that was missing:

> When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value.

In addition, it was added that the `removeNamedItem` method raises a NO_MODIFICATION_ALLOWED_ERR `DOMException` [p.20] when the `NamedNodeMap` [p.36] is readonly.

Section 1.2 Interface Node [p.28]

It was added to the description of the `nodeValue` attribute that setting it, when it is defined to be `null`, has no effect.

It was also added to the description of the `parentNode` attribute that `Entity` [p.51] and `Notation` [p.50] nodes do not have a parent.

It was also added to the description of the `NodeType` group:

> Numeric codes up to 200 are reserved to W3C for possible future use.

The following description has been removed:

> The content of the returned `NodeList` [p.35] 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.

The description of the NO_MODIFICATION_ALLOWED_ERR exception of the `insertBefore` method read:

> Raised if this node is readonly.

It now reads:

> Raised if this node is readonly or if the parent of the node being inserted is readonly.

The description of the `replaceChild` method was missing:

> If `newChild` is a `DocumentFragment` [p.23] object, `oldChild` is replaced by all of the `DocumentFragment` children, which are inserted in the same order. If the `newChild` is already in the tree, it is first removed.

The description of the NO_MODIFICATION_ALLOWED_ERR exception of the `replaceChild` method read:

> Raised if this node is readonly.

It now reads:

> Raised if this node or the parent of the new node is readonly.

The description of the `cloneNode` method was missing:

> Note that cloning an immutable subtree results in a mutable copy, but the children of an `EntityReference` [p.52] clone are *readonly* [p.128] . In addition, clones of unspecified `Attr` [p.42] nodes are specified. And, cloning `Document` [p.23] , `DocumentType` [p.49] , `Entity` [p.51] , and `Notation` [p.50] nodes is implementation dependent.

and also missing the following exceptions:

> NOT_SUPPORTED_ERR: Raised if this node is a of type `DOCUMENT_NODE`, `DOCUMENT_TYPE_NODE`, `ENTITY_NODE`, or `NOTATION_NODE` and the implementation does not support cloning this type of node.

Section 1.2 Interface NodeList [p.35]

The description was missing:

`NodeList` [p.35] objects in the DOM are *live* [p.18] .

Section 1.2 Interface Comment [p.48]

The first paragraph read:

> This represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'.

It now reads "This interface *inherits from CharacterData and* represents ...".

Section 1.2 Interface Text [p.47]

The first paragraph read:

> The `Text` [p.47] interface represents the textual content (termed character data in XML) of an `Element` [p.43] or `Attr` [p.42] .

It now reads "The Text interface *inherits from CharacterData and* represents ...".

The first paragraph read:

> If there is markup, it is parsed into a list of elements and `Text` [p.47] nodes that form the list of children of the element.

It has been clarified, now reads "into *the information items (elements, children comments, etc.)* and ...".

The last sentence of the second paragraph of the interface description read:

> The `normalize()` method on `Element` [p.43] merges any such adjacent `Text` [p.47] 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`.

However, since in cases where the document contains `CDATASections` [p.48] , the normalize operation alone may not be sufficient, since XPointers do not differentiate between `Text` [p.47] nodes and `CDATASection` [p.48] nodes, the last part of the sentence (after the semi-colon) was dropped.

The following sentence was added, for clarification purpose, to the description of the `splitText` method:

> When the `offset` is equal to the length of this node, the new `Text` [p.47] node has no data.

The description of the `slitText` method has been clarified:

> Breaks this `Text` [p.47] node into two `Text` at the specified `offset`, ...

It now reads "Breaks this node into two nodes ...".

The description `offset` parameter of the `splitText` method read:

The `offset` at which to split, starting from 0.

It now reads:

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

The description return value read:

The new `Text` [p.47] node.

It now reads:

The new node, of the same type as this node.

The exception INDEX_SIZE_ERR was said to be:

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

Instead this now reads:

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

Section 1.3 Extended Interfaces [p.48]

The following paragraph was missing:

A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.22] interface to determine whether they are supported or not. The feature string for all the interfaces listed in this section is "XML".

Section 1.3 Interface CDATASection [p.48]

The following note was added to the description:

Note: Because no markup is recognized within a `CDATASection` [p.48] , character numeric references cannot be used as an escape mechanism when serializing. Therefore, action needs to be taken when serializing a `CDATASection` with a character encoding where some of the contained characters cannot be represented. Failure to do so would not produce well-formed XML.
One potential solution in the serialization process is to end the CDATA section before the character, output the character using a character reference or entity reference, and open a new CDATA section for any further characters in the text node. Note, however, that some code conversion libraries at the time of writing do not return an error or exception when a character is missing from the encoding, making the task of ensuring that data is not corrupted on serialization more difficult.

Section 1.3. Interface Notation [p.50]

The first paragraph read:

they are therefore readonly.

A link to the glossary has been added for *readonly* [p.128] .

Section 1.3 Interface Entity [p.51]

The last sentence of the last paragraph was said to be:

> All the descendants of an `Entity` [p.51] node are readonly.

Instead, it now reads:

> `Entity` [p.51] nodes and all their descendants are *readonly* [p.128] .

Section 1.3 Interface EntityReference [p.52]

The following note was added to the description:

> As for `Entity` [p.51] nodes, `EntityReference` [p.52] nodes and all their descendants are *readonly* [p.128] .

A.3: Changes in the Document Object Model HTML

Section 2.1 Introduction [p.53]

The last sentence of the third paragraph was:

> Interoperability between implementations is only guaranteed for elements and attributes that are specified in these DTDs.

This seemed to imply interoperability is not guaranteed for the HTML 4.0 strict DTD, so it was changed to:

> Interoperability between implementations is only guaranteed for elements and attributes that are specified in the HTML 4.0 DTDs.

Section 2.5.1 Property Attributes [p.59]

"border" was given as an example of an open-value value list in the table. It was changed to "disabled".

Section 2.5.5 Interface HTMLBlockquoteElement

This interface was an error and was removed. The Interface HTMLQuoteElement [p.79] is used for both the Q and BLOCKQUOTE HTML elements.

Section 2.5.5 Interface HTMLInputElement [p.70]

The description of the `defaultValue` attribute now reads:

> value "Text", "File" or "Password", this represents the HTML value attribute of the element. The value of this attribute does not change if the contents of the corresponding form control in an interactive user agent changes. Changing this attribute, however, resets the contents of the form control. See the value attribute definition in HTML 4.0.

The description of the `value` attribute now reads:

> When the `type` attribute of the element has the value "Text", "File" or "Password", this represents the current contents of the corresponding form control in an interactive user agent. Changing this attribute changes the contents of the form control, but does not change the value of the HTML value attribute of the element. When the `type` attribute of the element has the value "Button", "Hidden", "Submit", "Reset", "Image", "Checkbox" or "Radio", this represents the HTML value attribute of the element. See the value attribute definition in HTML 4.0.

The description of the `defaultChecked` attribute now reads:

> When the `type` attribute of the element has the value "Checkbox" or "Radio", this represents the HTML checked attribute of the element. The value of this attribute does not change if the state of the corresponding form control in an interactive user agent changes. Changes to this attribute, however, resets the state of the form control. See the checked attribute definition in HTML 4.0.

The description of the `checked` attribute now reads:

> When the `type` attribute of the element has the value "Checkbox" or "Radio", this represents the current state of the corresponding form control in am interactive user agent. Changes to this attribute changes the state of the form control, but does not change the value of the HTML value attribute of the element.

Section 2.5.5 Interface HTMLOptionElement [p.69]

The `index` attribute was changed to readonly and the `selected` attribute to readwrite. It was also added that the `index` attribute starts from 0.

The description of the `defaultSelected` attribute was unclear, it now reads:

> Represents the value of the HTML selected attribute. The value of this attribute does not change if the state of the corresponding form control in an interactive user agent changes. Changing `defaultSelected`, however, resets the state of the form control. See the selected attribute definition in HTML 4.0.

In addition, its description was unclear, it now reads:

> Represents the current state of the corresponding form control in an interactive user agent. Changing this attribute changes the state of the form control, but does not change the value of the HTML selected attribute of the element.

Section 2.5.5 Interface HTMLSelectElement [p.66]

It was added that the value of the `type` attribute is the string "select-multiple" when the `multiple` attribute is `true` and the string "select-one" when `false`.

It was also added that the `index` attribute starts from 0.

The description of the `before` parameter of the `add` method read:

> The element to insert before, or `null` for the head of the list.

It was changed to:

> The element to insert before, or `null` for the tail of the list.

Finally, it was added that an NOT_FOUND_ERR `DOMException` [p.20] is raised when the `before` given to the `add` method is not a descendant of the `SELECT` element.

Section 2.5.5 Interface HTMLTableCellElement [p.99]

The `cellIndex` attribute was changed to readonly and it was added that it starts from 0.

Section 2.5.5 Interface HTMLTableElement [p.91]

The description of the `index` parameter of the `insertRow` and `deleteRow` methods was augmented with the following:

> This index starts from 0 and is relative to all the rows contained inside the table, regardless of section parentage.

In addition, the following was added to the description of the `insertRow` method:

> The new row is inserted immediately before and in the same section as the current `index`th row in the table. If there is no such row, the row is inserted following the one before in the table. Finally, when the table is empty the row is inserted into a `TBODY` which is created and inserted into the table.

Finally, it was added that an INDEX_SIZE_ERR `DOMException` [p.20] is raised when the `index` given to the `insertRow` method is greater than the number of rows, and when the `index` given to the `deleteRow` method is greater than or equal to the number of rows. In both case, the exception is also raised if the index is negative.

Section 2.5.5 Interface HTMLTableRowElement [p.97]

The `rowIndex`, `selectionRowIndex`, and `cells` attributes were changed to readonly. And it was added that these indexes start from 0.

It was added that the `index` parameter of the `insertCell` and `deleteCell` methods starts from 0.

The following sentence was added to the description of the `insertCell`:

> If `index` is equal to the number of cells, the new cell is appended.

In addition, it was added that an INDEX_SIZE_ERR `DOMException` [p.20] is raised when the `index` given to the `insertCell` method is greater than the number of cells and when the `index` given to the `deleteCell` method is greater than or equal to the number of cells. In both case, the exception is also raised if the index is negative.

Section 2.5.5 Interface HTMLTableSectionElement [p.96]

The description of the `index` parameter of the `insertRow` and `deleteRow` methods was augmented with the following:

> This index starts from 0 and is relative only to the rows contained inside this section, not all the rows in the table.

In addition, it was added that an INDEX_SIZE_ERR `DOMException` [p.20] is raised when the `index` given to the `insertRow` method is greater than the number of rows, and when the `index` given to the `deleteRow` method is greater than or equal to the number of rows. In both case, the exception is also raised if the index is negative.

Section 2.5.5 Interface HTMLTextArea [p.73]

The value of the `type` attribute is now defined to be the string "textarea".

The description of the `defaultValue` attribute was unclear, it now reads:

> Represents the contents of the element. The value of this attribute does not change if the contents of the corresponding form control in an interactive user agent changes. Changing this attribute, however, resets the contents of the form control.

The description of the `value` attribute was unclear, it now reads:

> Represents the current contents of the corresponding form control in an interactive user agent. Changing this attribute changes the contents of the form control, but does not change the contents of the element.

A.4: Changes in the Appendices

Appendix D (formerly C): IDL Definition [p.129]

The list of exception codes appeared twice and several pieces of information, such as the names and prefixes of the modules were missing.

We also added the following paragraph:

"Unfortunately the OMG IDL for the Document Object Model HTML is not compliant because of problems in the validator that was used to validate Level 1. The readOnly attribute on the HTMLInputElement and HTMLTextAreaElement interfaces, as well as the object attribute on the HTMLAppletElement interface, are not compliant with OMG IDL 2.2. The valueType attribute on the HTMLParamElement interface is not compliant with OMG IDL 2.3, which hadn't been released when DOM Level 1 was published."

Appendix E (formerly D): Java Language Binding [p.143]

The java source files did not contain the copyright notice appropriate for people to use them in their products. A few errors in the javadoc part of them were also fixed.

References [p.201]

The references were sorted by alphabetical order, severals references were added and updated.

Appendix E: Acknowledgements

The authors of this specification, members of the DOM Working Group, deserve much credit for their hard work: 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 Research and Software AG), Mike Champion (ArborText and Software AG), 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).

Thanks to all those who have helped to improve this specification by sending suggestions and corrections.

Thanks to Joe English, author of cost, which was used as the basis for producing DOM Level 1. Thanks also to Gavin Nicol, who wrote the scripts which run on top of cost. Arnaud Le Hors and Philippe Le Hégaret maintained the scripts.

For DOM Level 1 Second edition, we used Xerces as the basis DOM implementation and wish to thank the authors. Philippe Le Hégaret and Arnaud Le Hors wrote the Java programs which are the DOM application.

Thanks to Jan Kärrman, author of html2ps for helping so much in creating the Postscript version of the specification.

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.

16-bit unit
The base unit of a `DOMString` [p.19] . This indicates that indexing on a `DOMString` occurs in units of 16 bits. This must not be misunderstood to mean that a `DOMString` can store arbitrary 16-bit units. A `DOMString` is a character string encoded in UTF-16; this means that the restrictions of UTF-16 as well as the other relevant restrictions on character strings must be maintained. A single character, for example in the form of a numeric character reference, may correspond to one or two 16-bit units.
For more information, see [Unicode] and [ISO/IEC 10646].

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 [COM], 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 *Element Content* in XML [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 [CORBA]. 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."

DOM Level 0

The term "DOM Level 0" refers to a mix (not formally specified) of HTML document functionalities offered by Netscape Navigator version 3.0 and Microsoft Internet Explorer version 3.0. In some cases, attributes or methods have been included for reasons of backward compatibility with "DOM Level 0".

ECMAScript

The programming language defined by the ECMA-262 standard [ECMAScript]. As stated in the standard, the originating technology for ECMAScript was JavaScript [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. See *Logical Structures* in XML [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 the 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*, their child node lists are equivalent are equivalent as `NodeList` [p.35] objects, and their attributes are deeply equivalent.
Two `NodeList` [p.35] objects are *equivalent* if they have the same length, and the nodes corresponding by index are deeply equivalent.
Two `NamedNodeMap` [p.36] objects are *equivalent* if they have the same length, they have same collection of names, and the nodes corresponding by name in the maps are deeply equivalent.
Two `DocumentType` [p.49] nodes are *equivalent* if they are equivalent as nodes, have the same names, and have equivalent entities and attributes `NamedNodeMap` [p.36] objects.

information item

An information item is an abstract representation of some component of an XML document. See the [Infoset] for details.

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. [HTML3.2] [HTML4.0]

IDL

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

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.

readonly node

A *readonly node* is a node that is immutable. This means its list of children, its content, and its attributes, when it is an element, cannot be changed in any way. However, a readonly node can possibly be moved, when it is not itself contained in a readonly 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. *Well-Formed XML Documents* in XML [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 3.0 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. 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]

XML name

See *XML name* in the XML specification [XML].

Appendix B: IDL Definitions

This appendix contains the complete OMG IDL for the Level 1 Document Object Model definitions. The definitions are divided into Core [p.129] , HTML [p.133] .

The IDL files are also available as: http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/idl.zip

Unfortunately the OMG IDL for the Document Object Model HTML is not compliant because of problems in the validator that was used to validate Level 1. The `readOnly` attribute on the `HTMLInputElement` [p.70] and `HTMLTextAreaElement` [p.73] interfaces, as well as the `object` attribute on the `HTMLAppletElement` [p.88] interface, are not compliant with OMG IDL 2.2.

B.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.

dom.idl:

```
// File: dom.idl

#ifndef _DOM_IDL_
#define _DOM_IDL_

#pragma prefix "w3c.org"
module dom
{

  typedef sequence<unsigned short> DOMString;

  interface NodeList;
  interface NamedNodeMap;
  interface Document;

  exception DOMException {
    unsigned short   code;
  };
  // ExceptionCode
  const unsigned short      INDEX_SIZE_ERR                = 1;
  const unsigned short      DOMSTRING_SIZE_ERR            = 2;
  const unsigned short      HIERARCHY_REQUEST_ERR         = 3;
  const unsigned short      WRONG_DOCUMENT_ERR            = 4;
  const unsigned short      INVALID_CHARACTER_ERR         = 5;
  const unsigned short      NO_DATA_ALLOWED_ERR           = 6;
  const unsigned short      NO_MODIFICATION_ALLOWED_ERR   = 7;
  const unsigned short      NOT_FOUND_ERR                 = 8;
  const unsigned short      NOT_SUPPORTED_ERR             = 9;
  const unsigned short      INUSE_ATTRIBUTE_ERR           = 10;


  interface DOMImplementation {
    boolean            hasFeature(in DOMString feature,
```

```
                                in DOMString version);
};

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 DOMString        nodeName;
           attribute DOMString        nodeValue;
                                      // raises(DOMException) on setting
                                      // raises(DOMException) on retrieval

  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)
                                      raises(DOMException);
};

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

interface NamedNodeMap {
  Node                getNamedItem(in DOMString name);
  Node                setNamedItem(in Node arg)
                                      raises(DOMException);
```

```
  Node               removeNamedItem(in DOMString name)
                                    raises(DOMException);
  Node               item(in unsigned long index);
  readonly attribute unsigned long    length;
};

interface CharacterData : Node {
           attribute DOMString        data;
                                    // raises(DOMException) on setting
                                    // raises(DOMException) on retrieval

  readonly attribute unsigned long    length;
  DOMString          substringData(in unsigned long offset,
                                   in unsigned long count)
                                    raises(DOMException);
  void               appendData(in DOMString arg)
                                    raises(DOMException);
  void               insertData(in unsigned long offset,
                                in DOMString 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 DOMString arg)
                                    raises(DOMException);
};

interface Attr : Node {
  readonly attribute DOMString        name;
  readonly attribute boolean          specified;
  // Modified in DOM Level 1:
           attribute DOMString        value;
                                    // raises(DOMException) on setting

};

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

interface Text : CharacterData {
  Text               splitText(in unsigned long offset)
```

```
                                        raises(DOMException);
};

interface Comment : CharacterData {
};

interface CDATASection : Text {
};

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

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

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

interface EntityReference : Node {
};

interface ProcessingInstruction : Node {
  readonly attribute DOMString        target;
           attribute DOMString        data;
                                      // raises(DOMException) on setting

};

interface DocumentFragment : Node {
};

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

```
    NodeList           getElementsByTagName(in DOMString tagname);
  };
};

#endif // _DOM_IDL_
```

B.2: Document Object Model Level 1 HTML

html.idl:

```
// File: html.idl

#ifndef _HTML_IDL_
#define _HTML_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module html
{

  typedef dom::DOMString DOMString;
  typedef dom::Node Node;
  typedef dom::Document Document;
  typedef dom::Element Element;
  typedef dom::NodeList NodeList;

  interface HTMLElement;
  interface HTMLFormElement;
  interface HTMLTableCaptionElement;
  interface HTMLTableSectionElement;

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

  interface HTMLDocument : Document {
             attribute DOMString        title;
    readonly attribute DOMString        referrer;
    readonly attribute DOMString        domain;
    readonly attribute DOMString        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 DOMString        cookie;
    void               open();
    void               close();
    void               write(in DOMString text);
    void               writeln(in DOMString text);
    Element            getElementById(in DOMString elementId);
```

```
  NodeList           getElementsByName(in DOMString elementName);
};

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

interface HTMLHtmlElement : HTMLElement {
           attribute DOMString        version;
};

interface HTMLHeadElement : HTMLElement {
           attribute DOMString        profile;
};

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

interface HTMLTitleElement : HTMLElement {
           attribute DOMString        text;
};

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

interface HTMLBaseElement : HTMLElement {
           attribute DOMString        href;
           attribute DOMString        target;
};

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

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

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

interface HTMLFormElement : HTMLElement {
  readonly attribute HTMLCollection   elements;
  readonly attribute long             length;
           attribute DOMString        name;
           attribute DOMString        acceptCharset;
           attribute DOMString        action;
           attribute DOMString        enctype;
           attribute DOMString        method;
           attribute DOMString        target;
  void               submit();
  void               reset();
};

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

interface HTMLOptGroupElement : HTMLElement {
           attribute boolean          disabled;
           attribute DOMString        label;
};

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

```
};

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

interface HTMLTextAreaElement : HTMLElement {
           attribute DOMString        defaultValue;
  readonly attribute HTMLFormElement  form;
           attribute DOMString        accessKey;
           attribute long             cols;
           attribute boolean          disabled;
           attribute DOMString        name;
           attribute boolean          readOnly;
           attribute long             rows;
           attribute long             tabIndex;
  readonly attribute DOMString        type;
           attribute DOMString        value;
  void               blur();
  void               focus();
  void               select();
};

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

interface HTMLLabelElement : HTMLElement {
  readonly attribute HTMLFormElement  form;
           attribute DOMString        accessKey;
```

```
           attribute DOMString        htmlFor;
};

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

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

interface HTMLUListElement : HTMLElement {
           attribute boolean          compact;
           attribute DOMString        type;
};

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

interface HTMLDListElement : HTMLElement {
           attribute boolean          compact;
};

interface HTMLDirectoryElement : HTMLElement {
           attribute boolean          compact;
};

interface HTMLMenuElement : HTMLElement {
           attribute boolean          compact;
};

interface HTMLLIElement : HTMLElement {
           attribute DOMString        type;
           attribute long             value;
};

interface HTMLDivElement : HTMLElement {
           attribute DOMString        align;
};

interface HTMLParagraphElement : HTMLElement {
           attribute DOMString        align;
};

interface HTMLHeadingElement : HTMLElement {
           attribute DOMString        align;
};

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

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

interface HTMLBRElement : HTMLElement {
           attribute DOMString        clear;
};

interface HTMLBaseFontElement : HTMLElement {
           attribute DOMString        color;
           attribute DOMString        face;
           attribute DOMString        size;
};

interface HTMLFontElement : HTMLElement {
           attribute DOMString        color;
           attribute DOMString        face;
           attribute DOMString        size;
};

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

interface HTMLModElement : HTMLElement {
           attribute DOMString        cite;
           attribute DOMString        dateTime;
};

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

interface HTMLImageElement : HTMLElement {
           attribute DOMString        lowSrc;
           attribute DOMString        name;
           attribute DOMString        align;
           attribute DOMString        alt;
           attribute DOMString        border;
           attribute DOMString        height;
           attribute DOMString        hspace;
```

```
           attribute boolean          isMap;
           attribute DOMString        longDesc;
           attribute DOMString        src;
           attribute DOMString        useMap;
           attribute DOMString        vspace;
           attribute DOMString        width;
};

interface HTMLObjectElement : HTMLElement {
  readonly attribute HTMLFormElement  form;
           attribute DOMString        code;
           attribute DOMString        align;
           attribute DOMString        archive;
           attribute DOMString        border;
           attribute DOMString        codeBase;
           attribute DOMString        codeType;
           attribute DOMString        data;
           attribute boolean          declare;
           attribute DOMString        height;
           attribute DOMString        hspace;
           attribute DOMString        name;
           attribute DOMString        standby;
           attribute long             tabIndex;
           attribute DOMString        type;
           attribute DOMString        useMap;
           attribute DOMString        vspace;
           attribute DOMString        width;
};

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

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

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

interface HTMLAreaElement : HTMLElement {
           attribute DOMString        accessKey;
```

```
           attribute DOMString        alt;
           attribute DOMString        coords;
           attribute DOMString        href;
           attribute boolean          noHref;
           attribute DOMString        shape;
           attribute long             tabIndex;
           attribute DOMString        target;
};

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

interface HTMLTableElement : HTMLElement {
           attribute HTMLTableCaptionElement  caption;
           attribute HTMLTableSectionElement  tHead;
           attribute HTMLTableSectionElement  tFoot;
  readonly attribute HTMLCollection   rows;
  readonly attribute HTMLCollection   tBodies;
           attribute DOMString        align;
           attribute DOMString        bgColor;
           attribute DOMString        border;
           attribute DOMString        cellPadding;
           attribute DOMString        cellSpacing;
           attribute DOMString        frame;
           attribute DOMString        rules;
           attribute DOMString        summary;
           attribute DOMString        width;
  HTMLElement        createTHead();
  void               deleteTHead();
  HTMLElement        createTFoot();
  void               deleteTFoot();
  HTMLElement        createCaption();
  void               deleteCaption();
  HTMLElement        insertRow(in long index)
                                     raises(dom::DOMException);
  void               deleteRow(in long index)
                                     raises(dom::DOMException);
};

interface HTMLTableCaptionElement : HTMLElement {
           attribute DOMString        align;
};

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

```
};

interface HTMLTableSectionElement : HTMLElement {
           attribute DOMString        align;
           attribute DOMString        ch;
           attribute DOMString        chOff;
           attribute DOMString        vAlign;
  readonly attribute HTMLCollection   rows;
  HTMLElement        insertRow(in long index)
                                      raises(dom::DOMException);
  void               deleteRow(in long index)
                                      raises(dom::DOMException);
};

interface HTMLTableRowElement : HTMLElement {
  readonly attribute long             rowIndex;
  readonly attribute long             sectionRowIndex;
  readonly attribute HTMLCollection   cells;
           attribute DOMString        align;
           attribute DOMString        bgColor;
           attribute DOMString        ch;
           attribute DOMString        chOff;
           attribute DOMString        vAlign;
  HTMLElement        insertCell(in long index)
                                      raises(dom::DOMException);
  void               deleteCell(in long index)
                                      raises(dom::DOMException);
};

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

interface HTMLFrameSetElement : HTMLElement {
           attribute DOMString        cols;
           attribute DOMString        rows;
};

interface HTMLFrameElement : HTMLElement {
           attribute DOMString        frameBorder;
           attribute DOMString        longDesc;
           attribute DOMString        marginHeight;
           attribute DOMString        marginWidth;
```

```
            attribute DOMString        name;
            attribute boolean          noResize;
            attribute DOMString        scrolling;
            attribute DOMString        src;
  };

  interface HTMLIFrameElement : HTMLElement {
            attribute DOMString        align;
            attribute DOMString        frameBorder;
            attribute DOMString        height;
            attribute DOMString        longDesc;
            attribute DOMString        marginHeight;
            attribute DOMString        marginWidth;
            attribute DOMString        name;
            attribute DOMString        scrolling;
            attribute DOMString        src;
            attribute DOMString        width;
  };
};

#endif // _HTML_IDL_
```

Appendix C: Java Language Binding

This appendix contains the complete Java binding for the Level 1 Document Object Model. The definitions are divided into Core [p.143] , HTML [p.149] .

The Java files are also available as
http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/java-binding.zip

C.1: Document Object Model Level 1 Core

org/w3c/dom/DOMException.java:

```
package org.w3c.dom;

public class DOMException extends RuntimeException {
    public DOMException(short code, String message) {
       super(message);
       this.code = code;
    }
    public short   code;
    // ExceptionCode
    public static final short INDEX_SIZE_ERR            = 1;
    public static final short DOMSTRING_SIZE_ERR        = 2;
    public static final short HIERARCHY_REQUEST_ERR     = 3;
    public static final short WRONG_DOCUMENT_ERR        = 4;
    public static final short INVALID_CHARACTER_ERR     = 5;
    public static final short NO_DATA_ALLOWED_ERR       = 6;
    public static final short NO_MODIFICATION_ALLOWED_ERR = 7;
    public static final short NOT_FOUND_ERR             = 8;
    public static final short NOT_SUPPORTED_ERR         = 9;
    public static final short INUSE_ATTRIBUTE_ERR       = 10;

}
```

org/w3c/dom/DOMImplementation.java:

```
package org.w3c.dom;

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

}
```

org/w3c/dom/DocumentFragment.java:

```
package org.w3c.dom;

public interface DocumentFragment extends Node {
}
```

org/w3c/dom/Document.java:

```
package org.w3c.dom;

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 Attr createAttribute(String name)
                                throws DOMException;

    public EntityReference createEntityReference(String name)
                                                 throws DOMException;

    public NodeList getElementsByTagName(String tagname);

}
```

org/w3c/dom/Node.java:

```
package org.w3c.dom;

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()
                                        throws DOMException;
    public void setNodeValue(String nodeValue)
                                        throws DOMException;

    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)
                          throws DOMException;

}
```

org/w3c/dom/NodeList.java:

```
package org.w3c.dom;

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

    public int getLength();

}
```

org/w3c/dom/NamedNodeMap.java:

```
package org.w3c.dom;

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();

}
```

org/w3c/dom/CharacterData.java:

```
package org.w3c.dom;

public interface CharacterData extends Node {
    public String getData()
                    throws DOMException;
    public void setData(String data)
                    throws DOMException;

    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;

}
```

org/w3c/dom/Attr.java:

```
package org.w3c.dom;

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

    public boolean getSpecified();

    public String getValue();
    public void setValue(String value)
                         throws DOMException;

}
```

org/w3c/dom/Element.java:

```
package org.w3c.dom;

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 Attr getAttributeNode(String name);

    public Attr setAttributeNode(Attr newAttr)
                                 throws DOMException;

    public Attr removeAttributeNode(Attr oldAttr)
                                    throws DOMException;

    public NodeList getElementsByTagName(String name);

    public void normalize();

}
```

org/w3c/dom/Text.java:

```
package org.w3c.dom;

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

}
```

org/w3c/dom/Comment.java:

```
package org.w3c.dom;

public interface Comment extends CharacterData {
}
```

org/w3c/dom/CDATASection.java:

```
package org.w3c.dom;

public interface CDATASection extends Text {
}
```

org/w3c/dom/DocumentType.java:

```
package org.w3c.dom;

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

    public NamedNodeMap getEntities();

    public NamedNodeMap getNotations();

}
```

org/w3c/dom/Notation.java:

```
package org.w3c.dom;

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

    public String getSystemId();

}
```

org/w3c/dom/Entity.java:

```
package org.w3c.dom;

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

    public String getSystemId();

    public String getNotationName();

}
```

org/w3c/dom/EntityReference.java:

```
package org.w3c.dom;

public interface EntityReference extends Node {
}
```

org/w3c/dom/ProcessingInstruction.java:

```
package org.w3c.dom;

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

    public String getData();
    public void setData(String data)
                         throws DOMException;

}
```

C.2: Document Object Model Level 1 HTML

org/w3c/dom/html/HTMLCollection.java:

```
package org.w3c.dom.html;

import org.w3c.dom.Node;

public interface HTMLCollection {
    public int getLength();

    public Node item(int index);

    public Node namedItem(String name);

}
```

org/w3c/dom/html/HTMLDocument.java:

```
package org.w3c.dom.html;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;

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

    public String getReferrer();

    public String getDomain();
```

```
    public String getURL();

    public HTMLElement getBody();
    public void setBody(HTMLElement body);

    public HTMLCollection getImages();

    public HTMLCollection getApplets();

    public HTMLCollection getLinks();

    public HTMLCollection getForms();

    public HTMLCollection getAnchors();

    public String getCookie();
    public void setCookie(String cookie);

    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);

}
```

org/w3c/dom/html/HTMLElement.java:

```
package org.w3c.dom.html;

import org.w3c.dom.Element;

public interface HTMLElement extends Element {
    public String getId();
    public void setId(String id);

    public String getTitle();
    public void setTitle(String title);

    public String getLang();
    public void setLang(String lang);

    public String getDir();
    public void setDir(String dir);

    public String getClassName();
    public void setClassName(String className);

}
```

org/w3c/dom/html/HTMLHtmlElement.java:

```
package org.w3c.dom.html;

public interface HTMLHtmlElement extends HTMLElement {
    public String getVersion();
    public void setVersion(String version);

}
```

org/w3c/dom/html/HTMLHeadElement.java:

```
package org.w3c.dom.html;

public interface HTMLHeadElement extends HTMLElement {
    public String getProfile();
    public void setProfile(String profile);

}
```

org/w3c/dom/html/HTMLLinkElement.java:

```
package org.w3c.dom.html;

public interface HTMLLinkElement extends HTMLElement {
    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public String getCharset();
    public void setCharset(String charset);

    public String getHref();
    public void setHref(String href);

    public String getHreflang();
    public void setHreflang(String hreflang);

    public String getMedia();
    public void setMedia(String media);

    public String getRel();
    public void setRel(String rel);

    public String getRev();
    public void setRev(String rev);

    public String getTarget();
    public void setTarget(String target);

    public String getType();
    public void setType(String type);

}
```

org/w3c/dom/html/HTMLTitleElement.java:

```
package org.w3c.dom.html;

public interface HTMLTitleElement extends HTMLElement {
    public String getText();
    public void setText(String text);

}
```

org/w3c/dom/html/HTMLMetaElement.java:

```
package org.w3c.dom.html;

public interface HTMLMetaElement extends HTMLElement {
    public String getContent();
    public void setContent(String content);

    public String getHttpEquiv();
    public void setHttpEquiv(String httpEquiv);

    public String getName();
    public void setName(String name);

    public String getScheme();
    public void setScheme(String scheme);

}
```

org/w3c/dom/html/HTMLBaseElement.java:

```
package org.w3c.dom.html;

public interface HTMLBaseElement extends HTMLElement {
    public String getHref();
    public void setHref(String href);

    public String getTarget();
    public void setTarget(String target);

}
```

org/w3c/dom/html/HTMLIsIndexElement.java:

```
package org.w3c.dom.html;

public interface HTMLIsIndexElement extends HTMLElement {
    public HTMLFormElement getForm();

    public String getPrompt();
    public void setPrompt(String prompt);

}
```

org/w3c/dom/html/HTMLStyleElement.java:

```
package org.w3c.dom.html;

public interface HTMLStyleElement extends HTMLElement {
    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public String getMedia();
    public void setMedia(String media);

    public String getType();
    public void setType(String type);

}
```

org/w3c/dom/html/HTMLBodyElement.java:

```
package org.w3c.dom.html;

public interface HTMLBodyElement extends HTMLElement {
    public String getALink();
    public void setALink(String aLink);

    public String getBackground();
    public void setBackground(String background);

    public String getBgColor();
    public void setBgColor(String bgColor);

    public String getLink();
    public void setLink(String link);

    public String getText();
    public void setText(String text);

    public String getVLink();
    public void setVLink(String vLink);

}
```

org/w3c/dom/html/HTMLFormElement.java:

```
package org.w3c.dom.html;

public interface HTMLFormElement extends HTMLElement {
    public HTMLCollection getElements();

    public int getLength();

    public String getName();
    public void setName(String name);

    public String getAcceptCharset();
    public void setAcceptCharset(String acceptCharset);
```

```
    public String getAction();
    public void setAction(String action);

    public String getEnctype();
    public void setEnctype(String enctype);

    public String getMethod();
    public void setMethod(String method);

    public String getTarget();
    public void setTarget(String target);

    public void submit();

    public void reset();

}
```

org/w3c/dom/html/HTMLSelectElement.java:

```
package org.w3c.dom.html;

import org.w3c.dom.DOMException;

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

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

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

    public int getLength();

    public HTMLFormElement getForm();

    public HTMLCollection getOptions();

    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public boolean getMultiple();
    public void setMultiple(boolean multiple);

    public String getName();
    public void setName(String name);

    public int getSize();
    public void setSize(int size);

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

    public void add(HTMLElement element,
```

```
                    HTMLElement before)
                    throws DOMException;

    public void remove(int index);

    public void blur();

    public void focus();

}
```

org/w3c/dom/html/HTMLOptGroupElement.java:

```
package org.w3c.dom.html;

public interface HTMLOptGroupElement extends HTMLElement {
    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public String getLabel();
    public void setLabel(String label);

}
```

org/w3c/dom/html/HTMLOptionElement.java:

```
package org.w3c.dom.html;

public interface HTMLOptionElement extends HTMLElement {
    public HTMLFormElement getForm();

    public boolean getDefaultSelected();
    public void setDefaultSelected(boolean defaultSelected);

    public String getText();

    public int getIndex();

    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public String getLabel();
    public void setLabel(String label);

    public boolean getSelected();
    public void setSelected(boolean selected);

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

}
```

org/w3c/dom/html/HTMLInputElement.java:

```
package org.w3c.dom.html;

public interface HTMLInputElement extends HTMLElement {
    public String getDefaultValue();
    public void setDefaultValue(String defaultValue);

    public boolean getDefaultChecked();
    public void setDefaultChecked(boolean defaultChecked);

    public HTMLFormElement getForm();

    public String getAccept();
    public void setAccept(String accept);

    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public String getAlign();
    public void setAlign(String align);

    public String getAlt();
    public void setAlt(String alt);

    public boolean getChecked();
    public void setChecked(boolean checked);

    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public int getMaxLength();
    public void setMaxLength(int maxLength);

    public String getName();
    public void setName(String name);

    public boolean getReadOnly();
    public void setReadOnly(boolean readOnly);

    public String getSize();
    public void setSize(String size);

    public String getSrc();
    public void setSrc(String src);

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

    public String getType();

    public String getUseMap();
    public void setUseMap(String useMap);

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

```
    public void blur();

    public void focus();

    public void select();

    public void click();

}
```

org/w3c/dom/html/HTMLTextAreaElement.java:

```
package org.w3c.dom.html;

public interface HTMLTextAreaElement extends HTMLElement {
    public String getDefaultValue();
    public void setDefaultValue(String defaultValue);

    public HTMLFormElement getForm();

    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public int getCols();
    public void setCols(int cols);

    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public String getName();
    public void setName(String name);

    public boolean getReadOnly();
    public void setReadOnly(boolean readOnly);

    public int getRows();
    public void setRows(int rows);

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

    public String getType();

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

    public void blur();

    public void focus();

    public void select();

}
```

org/w3c/dom/html/HTMLButtonElement.java:

```
package org.w3c.dom.html;

public interface HTMLButtonElement extends HTMLElement {
    public HTMLFormElement getForm();

    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public String getName();
    public void setName(String name);

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

    public String getType();

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

}
```

org/w3c/dom/html/HTMLLabelElement.java:

```
package org.w3c.dom.html;

public interface HTMLLabelElement extends HTMLElement {
    public HTMLFormElement getForm();

    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public String getHtmlFor();
    public void setHtmlFor(String htmlFor);

}
```

org/w3c/dom/html/HTMLFieldSetElement.java:

```
package org.w3c.dom.html;

public interface HTMLFieldSetElement extends HTMLElement {
    public HTMLFormElement getForm();

}
```

org/w3c/dom/html/HTMLLegendElement.java:

```
package org.w3c.dom.html;

public interface HTMLLegendElement extends HTMLElement {
    public HTMLFormElement getForm();

    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public String getAlign();
    public void setAlign(String align);

}
```

org/w3c/dom/html/HTMLUListElement.java:

```
package org.w3c.dom.html;

public interface HTMLUListElement extends HTMLElement {
    public boolean getCompact();
    public void setCompact(boolean compact);

    public String getType();
    public void setType(String type);

}
```

org/w3c/dom/html/HTMLOListElement.java:

```
package org.w3c.dom.html;

public interface HTMLOListElement extends HTMLElement {
    public boolean getCompact();
    public void setCompact(boolean compact);

    public int getStart();
    public void setStart(int start);

    public String getType();
    public void setType(String type);

}
```

org/w3c/dom/html/HTMLDListElement.java:

```
package org.w3c.dom.html;

public interface HTMLDListElement extends HTMLElement {
    public boolean getCompact();
    public void setCompact(boolean compact);

}
```

org/w3c/dom/html/HTMLDirectoryElement.java:

```
package org.w3c.dom.html;

public interface HTMLDirectoryElement extends HTMLElement {
    public boolean getCompact();
    public void setCompact(boolean compact);

}
```

org/w3c/dom/html/HTMLMenuElement.java:

```
package org.w3c.dom.html;

public interface HTMLMenuElement extends HTMLElement {
    public boolean getCompact();
    public void setCompact(boolean compact);

}
```

org/w3c/dom/html/HTMLLIElement.java:

```
package org.w3c.dom.html;

public interface HTMLLIElement extends HTMLElement {
    public String getType();
    public void setType(String type);

    public int getValue();
    public void setValue(int value);

}
```

org/w3c/dom/html/HTMLDivElement.java:

```
package org.w3c.dom.html;

public interface HTMLDivElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

}
```

org/w3c/dom/html/HTMLParagraphElement.java:

```
package org.w3c.dom.html;

public interface HTMLParagraphElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

}
```

org/w3c/dom/html/HTMLHeadingElement.java:

```
package org.w3c.dom.html;

public interface HTMLHeadingElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

}
```

org/w3c/dom/html/HTMLQuoteElement.java:

```
package org.w3c.dom.html;

public interface HTMLQuoteElement extends HTMLElement {
    public String getCite();
    public void setCite(String cite);

}
```

org/w3c/dom/html/HTMLPreElement.java:

```
package org.w3c.dom.html;

public interface HTMLPreElement extends HTMLElement {
    public int getWidth();
    public void setWidth(int width);

}
```

org/w3c/dom/html/HTMLBRElement.java:

```
package org.w3c.dom.html;

public interface HTMLBRElement extends HTMLElement {
    public String getClear();
    public void setClear(String clear);

}
```

org/w3c/dom/html/HTMLBaseFontElement.java:

```
package org.w3c.dom.html;

public interface HTMLBaseFontElement extends HTMLElement {
    public String getColor();
    public void setColor(String color);

    public String getFace();
    public void setFace(String face);
```

```
    public String getSize();
    public void setSize(String size);

}
```

org/w3c/dom/html/HTMLFontElement.java:

```
package org.w3c.dom.html;

public interface HTMLFontElement extends HTMLElement {
    public String getColor();
    public void setColor(String color);

    public String getFace();
    public void setFace(String face);

    public String getSize();
    public void setSize(String size);

}
```

org/w3c/dom/html/HTMLHRElement.java:

```
package org.w3c.dom.html;

public interface HTMLHRElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

    public boolean getNoShade();
    public void setNoShade(boolean noShade);

    public String getSize();
    public void setSize(String size);

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

}
```

org/w3c/dom/html/HTMLModElement.java:

```
package org.w3c.dom.html;

public interface HTMLModElement extends HTMLElement {
    public String getCite();
    public void setCite(String cite);

    public String getDateTime();
    public void setDateTime(String dateTime);

}
```

org/w3c/dom/html/HTMLAnchorElement.java:

```
package org.w3c.dom.html;

public interface HTMLAnchorElement extends HTMLElement {
    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public String getCharset();
    public void setCharset(String charset);

    public String getCoords();
    public void setCoords(String coords);

    public String getHref();
    public void setHref(String href);

    public String getHreflang();
    public void setHreflang(String hreflang);

    public String getName();
    public void setName(String name);

    public String getRel();
    public void setRel(String rel);

    public String getRev();
    public void setRev(String rev);

    public String getShape();
    public void setShape(String shape);

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

    public String getTarget();
    public void setTarget(String target);

    public String getType();
    public void setType(String type);

    public void blur();

    public void focus();

}
```

org/w3c/dom/html/HTMLImageElement.java:

```
package org.w3c.dom.html;

public interface HTMLImageElement extends HTMLElement {
    public String getLowSrc();
    public void setLowSrc(String lowSrc);

    public String getName();
```

```
    public void setName(String name);

    public String getAlign();
    public void setAlign(String align);

    public String getAlt();
    public void setAlt(String alt);

    public String getBorder();
    public void setBorder(String border);

    public String getHeight();
    public void setHeight(String height);

    public String getHspace();
    public void setHspace(String hspace);

    public boolean getIsMap();
    public void setIsMap(boolean isMap);

    public String getLongDesc();
    public void setLongDesc(String longDesc);

    public String getSrc();
    public void setSrc(String src);

    public String getUseMap();
    public void setUseMap(String useMap);

    public String getVspace();
    public void setVspace(String vspace);

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

}
```

org/w3c/dom/html/HTMLObjectElement.java:

```
package org.w3c.dom.html;

public interface HTMLObjectElement extends HTMLElement {
    public HTMLFormElement getForm();

    public String getCode();
    public void setCode(String code);

    public String getAlign();
    public void setAlign(String align);

    public String getArchive();
    public void setArchive(String archive);

    public String getBorder();
    public void setBorder(String border);
```

```
    public String getCodeBase();
    public void setCodeBase(String codeBase);

    public String getCodeType();
    public void setCodeType(String codeType);

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

    public boolean getDeclare();
    public void setDeclare(boolean declare);

    public String getHeight();
    public void setHeight(String height);

    public String getHspace();
    public void setHspace(String hspace);

    public String getName();
    public void setName(String name);

    public String getStandby();
    public void setStandby(String standby);

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

    public String getType();
    public void setType(String type);

    public String getUseMap();
    public void setUseMap(String useMap);

    public String getVspace();
    public void setVspace(String vspace);

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

}
```

org/w3c/dom/html/HTMLParamElement.java:

```
package org.w3c.dom.html;

public interface HTMLParamElement extends HTMLElement {
    public String getName();
    public void setName(String name);

    public String getType();
    public void setType(String type);

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

```
    public String getValueType();
    public void setValueType(String valueType);

}
```

org/w3c/dom/html/HTMLAppletElement.java:

```
package org.w3c.dom.html;

public interface HTMLAppletElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

    public String getAlt();
    public void setAlt(String alt);

    public String getArchive();
    public void setArchive(String archive);

    public String getCode();
    public void setCode(String code);

    public String getCodeBase();
    public void setCodeBase(String codeBase);

    public String getHeight();
    public void setHeight(String height);

    public String getHspace();
    public void setHspace(String hspace);

    public String getName();
    public void setName(String name);

    public String getObject();
    public void setObject(String object);

    public String getVspace();
    public void setVspace(String vspace);

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

}
```

org/w3c/dom/html/HTMLMapElement.java:

```
package org.w3c.dom.html;

public interface HTMLMapElement extends HTMLElement {
    public HTMLCollection getAreas();

    public String getName();
    public void setName(String name);

}
```

org/w3c/dom/html/HTMLAreaElement.java:

```
package org.w3c.dom.html;

public interface HTMLAreaElement extends HTMLElement {
    public String getAccessKey();
    public void setAccessKey(String accessKey);

    public String getAlt();
    public void setAlt(String alt);

    public String getCoords();
    public void setCoords(String coords);

    public String getHref();
    public void setHref(String href);

    public boolean getNoHref();
    public void setNoHref(boolean noHref);

    public String getShape();
    public void setShape(String shape);

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

    public String getTarget();
    public void setTarget(String target);

}
```

org/w3c/dom/html/HTMLScriptElement.java:

```
package org.w3c.dom.html;

public interface HTMLScriptElement extends HTMLElement {
    public String getText();
    public void setText(String text);

    public String getHtmlFor();
    public void setHtmlFor(String htmlFor);

    public String getEvent();
    public void setEvent(String event);

    public String getCharset();
    public void setCharset(String charset);

    public boolean getDefer();
    public void setDefer(boolean defer);

    public String getSrc();
    public void setSrc(String src);
```

```
    public String getType();
    public void setType(String type);

}
```

org/w3c/dom/html/HTMLTableElement.java:

```
package org.w3c.dom.html;

import org.w3c.dom.DOMException;

public interface HTMLTableElement extends HTMLElement {
    public HTMLTableCaptionElement getCaption();
    public void setCaption(HTMLTableCaptionElement caption);

    public HTMLTableSectionElement getTHead();
    public void setTHead(HTMLTableSectionElement tHead);

    public HTMLTableSectionElement getTFoot();
    public void setTFoot(HTMLTableSectionElement tFoot);

    public HTMLCollection getRows();

    public HTMLCollection getTBodies();

    public String getAlign();
    public void setAlign(String align);

    public String getBgColor();
    public void setBgColor(String bgColor);

    public String getBorder();
    public void setBorder(String border);

    public String getCellPadding();
    public void setCellPadding(String cellPadding);

    public String getCellSpacing();
    public void setCellSpacing(String cellSpacing);

    public String getFrame();
    public void setFrame(String frame);

    public String getRules();
    public void setRules(String rules);

    public String getSummary();
    public void setSummary(String summary);

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

    public HTMLElement createTHead();

    public void deleteTHead();
```

```
    public HTMLElement createTFoot();

    public void deleteTFoot();

    public HTMLElement createCaption();

    public void deleteCaption();

    public HTMLElement insertRow(int index)
                                 throws DOMException;

    public void deleteRow(int index)
                          throws DOMException;

}
```

org/w3c/dom/html/HTMLTableCaptionElement.java:

```
package org.w3c.dom.html;

public interface HTMLTableCaptionElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

}
```

org/w3c/dom/html/HTMLTableColElement.java:

```
package org.w3c.dom.html;

public interface HTMLTableColElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

    public String getCh();
    public void setCh(String ch);

    public String getChOff();
    public void setChOff(String chOff);

    public int getSpan();
    public void setSpan(int span);

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

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

}
```

org/w3c/dom/html/HTMLTableSectionElement.java:

```
package org.w3c.dom.html;

import org.w3c.dom.DOMException;

public interface HTMLTableSectionElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

    public String getCh();
    public void setCh(String ch);

    public String getChOff();
    public void setChOff(String chOff);

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

    public HTMLCollection getRows();

    public HTMLElement insertRow(int index)
                                 throws DOMException;

    public void deleteRow(int index)
                          throws DOMException;

}
```

org/w3c/dom/html/HTMLTableRowElement.java:

```
package org.w3c.dom.html;

import org.w3c.dom.DOMException;

public interface HTMLTableRowElement extends HTMLElement {
    public int getRowIndex();

    public int getSectionRowIndex();

    public HTMLCollection getCells();

    public String getAlign();
    public void setAlign(String align);

    public String getBgColor();
    public void setBgColor(String bgColor);

    public String getCh();
    public void setCh(String ch);

    public String getChOff();
    public void setChOff(String chOff);

    public String getVAlign();
    public void setVAlign(String vAlign);
```

```
    public HTMLElement insertCell(int index)
                                 throws DOMException;

    public void deleteCell(int index)
                           throws DOMException;

}
```

org/w3c/dom/html/HTMLTableCellElement.java:

```
package org.w3c.dom.html;

public interface HTMLTableCellElement extends HTMLElement {
    public int getCellIndex();

    public String getAbbr();
    public void setAbbr(String abbr);

    public String getAlign();
    public void setAlign(String align);

    public String getAxis();
    public void setAxis(String axis);

    public String getBgColor();
    public void setBgColor(String bgColor);

    public String getCh();
    public void setCh(String ch);

    public String getChOff();
    public void setChOff(String chOff);

    public int getColSpan();
    public void setColSpan(int colSpan);

    public String getHeaders();
    public void setHeaders(String headers);

    public String getHeight();
    public void setHeight(String height);

    public boolean getNoWrap();
    public void setNoWrap(boolean noWrap);

    public int getRowSpan();
    public void setRowSpan(int rowSpan);

    public String getScope();
    public void setScope(String scope);

    public String getVAlign();
    public void setVAlign(String vAlign);
```

```
    public String getWidth();
    public void setWidth(String width);

}
```

org/w3c/dom/html/HTMLFrameSetElement.java:

```
package org.w3c.dom.html;

public interface HTMLFrameSetElement extends HTMLElement {
    public String getCols();
    public void setCols(String cols);

    public String getRows();
    public void setRows(String rows);

}
```

org/w3c/dom/html/HTMLFrameElement.java:

```
package org.w3c.dom.html;

public interface HTMLFrameElement extends HTMLElement {
    public String getFrameBorder();
    public void setFrameBorder(String frameBorder);

    public String getLongDesc();
    public void setLongDesc(String longDesc);

    public String getMarginHeight();
    public void setMarginHeight(String marginHeight);

    public String getMarginWidth();
    public void setMarginWidth(String marginWidth);

    public String getName();
    public void setName(String name);

    public boolean getNoResize();
    public void setNoResize(boolean noResize);

    public String getScrolling();
    public void setScrolling(String scrolling);

    public String getSrc();
    public void setSrc(String src);

}
```

org/w3c/dom/html/HTMLIFrameElement.java:

```
package org.w3c.dom.html;

public interface HTMLIFrameElement extends HTMLElement {
    public String getAlign();
    public void setAlign(String align);

    public String getFrameBorder();
    public void setFrameBorder(String frameBorder);

    public String getHeight();
    public void setHeight(String height);

    public String getLongDesc();
    public void setLongDesc(String longDesc);

    public String getMarginHeight();
    public void setMarginHeight(String marginHeight);

    public String getMarginWidth();
    public void setMarginWidth(String marginWidth);

    public String getName();
    public void setName(String name);

    public String getScrolling();
    public void setScrolling(String scrolling);

    public String getSrc();
    public void setSrc(String src);

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

}
```

Appendix D: 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 [p.175] and HTML [p.181] .

D.1: Document Object Model Level 1 Core

Object **DOMImplementation**

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

hasFeature(feature, version)

This method returns a **Boolean**.
The **feature** parameter is of type **String**.
The **version** parameter is of type **String**.

Object **DocumentFragment**

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

Object **Document**

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

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

doctype

This read-only property is a **DocumentType** object.

implementation

This read-only property is a **DOMImplementation** object.

documentElement

This read-only property is a **Element** object.

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

createElement(tagName)

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

createDocumentFragment()

This method returns a **DocumentFragment** object.

createTextNode(data)

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

createComment(data)

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

createCDATASection(data)

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

createProcessingInstruction(target, data)

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

createAttribute(name)
This method returns a **Attr** object.
The **name** parameter is of type **String**.
createEntityReference(name)
This method returns a **EntityReference** object.
The **name** parameter is of type **String**.
getElementsByTagName(tagname)
This method returns a **NodeList** object.
The **tagname** parameter is of type **String**.

Prototype Object **Node**

The **Node** class has the following constants:

Node.ELEMENT_NODE
This constant is of type **Number** and its value is **1**.
Node.ATTRIBUTE_NODE
This constant is of type **Number** and its value is **2**.
Node.TEXT_NODE
This constant is of type **Number** and its value is **3**.
Node.CDATA_SECTION_NODE
This constant is of type **Number** and its value is **4**.
Node.ENTITY_REFERENCE_NODE
This constant is of type **Number** and its value is **5**.
Node.ENTITY_NODE
This constant is of type **Number** and its value is **6**.
Node.PROCESSING_INSTRUCTION_NODE
This constant is of type **Number** and its value is **7**.
Node.COMMENT_NODE
This constant is of type **Number** and its value is **8**.
Node.DOCUMENT_NODE
This constant is of type **Number** and its value is **9**.
Node.DOCUMENT_TYPE_NODE
This constant is of type **Number** and its value is **10**.
Node.DOCUMENT_FRAGMENT_NODE
This constant is of type **Number** and its value is **11**.
Node.NOTATION_NODE
This constant is of type **Number** and its value is **12**.

Object **Node**

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

nodeName
This read-only property is of type **String**.
nodeValue
This property is of type **String**.
nodeType
This read-only property is of type **Number**.
parentNode
This read-only property is a **Node** object.

childNodes
This read-only property is a **NodeList** object.
firstChild
This read-only property is a **Node** object.
lastChild
This read-only property is a **Node** object.
previousSibling
This read-only property is a **Node** object.
nextSibling
This read-only property is a **Node** object.
attributes
This read-only property is a **NamedNodeMap** object.
ownerDocument
This read-only property is a **Document** object.

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

insertBefore(newChild, refChild)
This method returns a **Node** object.
The **newChild** parameter is a **Node** object.
The **refChild** parameter is a **Node** object.
replaceChild(newChild, oldChild)
This method returns a **Node** object.
The **newChild** parameter is a **Node** object.
The **oldChild** parameter is a **Node** object.
removeChild(oldChild)
This method returns a **Node** object.
The **oldChild** parameter is a **Node** object.
appendChild(newChild)
This method returns a **Node** object.
The **newChild** parameter is a **Node** object.
hasChildNodes()
This method returns a **Boolean**.
cloneNode(deep)
This method returns a **Node** object.
The **deep** parameter is of type **Boolean**.

Object **NodeList**

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

length
This read-only property is of type **Number**.

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

item(index)
This method returns a **Node** object.
The **index** parameter is of type **Number**.
Note: This object can also be dereferenced using square bracket notation (e.g. obj[1]). Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

Object **NamedNodeMap**

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

length

This read-only property is of type **Number**.

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

getNamedItem(name)

This method returns a **Node** object.

The **name** parameter is of type **String**.

setNamedItem(arg)

This method returns a **Node** object.

The **arg** parameter is a **Node** object.

removeNamedItem(name)

This method returns a **Node** object.

The **name** parameter is of type **String**.

item(index)

This method returns a **Node** object.

The **index** parameter is of type **Number**.

Note: This object can also be dereferenced using square bracket notation (e.g. obj[1]). Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

Object **CharacterData**

CharacterData has the all the properties and methods of the **Node** object 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 read-only property is of type **Number**.

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

substringData(offset, count)

This method returns a **String**.

The **offset** parameter is of type **Number**.

The **count** parameter is of type **Number**.

appendData(arg)

This method has no return value.

The **arg** parameter is of type **String**.

insertData(offset, arg)

This method has no return value.

The **offset** parameter is of type **Number**.

The **arg** parameter is of type **String**.

deleteData(offset, count)

This method has no return value.

The **offset** parameter is of type **Number**.

The **count** parameter is of type **Number**.

replaceData(offset, count, arg)

This method has no return value.

The **offset** parameter is of type **Number**.
The **count** parameter is of type **Number**.
The **arg** parameter is of type **String**.

Object **Attr**

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

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

name

This read-only property is of type **String**.

specified

This read-only property is of type **Boolean**.

value

This property is of type **String**.

Object **Element**

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

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

tagName

This read-only property is of type **String**.

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

getAttribute(name)

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

setAttribute(name, value)

This method has no return value.
The **name** parameter is of type **String**.
The **value** parameter is of type **String**.

removeAttribute(name)

This method has no return value.
The **name** parameter is of type **String**.

getAttributeNode(name)

This method returns a **Attr** object.
The **name** parameter is of type **String**.

setAttributeNode(newAttr)

This method returns a **Attr** object.
The **newAttr** parameter is a **Attr** object.

removeAttributeNode(oldAttr)

This method returns a **Attr** object.
The **oldAttr** parameter is a **Attr** object.

getElementsByTagName(name)

This method returns a **NodeList** object.
The **name** parameter is of type **String**.

normalize()

This method has no return value.

Object **Text**

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

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

splitText(offset)

This method returns a **Text** object.

The **offset** parameter is of type **Number**.

Object **Comment**

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

Object **CDATASection**

CDATASection has the all the properties and methods of the **Text** object as well as the properties and methods defined below.

Object **DocumentType**

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

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

name

This read-only property is of type **String**.

entities

This read-only property is a **NamedNodeMap** object.

notations

This read-only property is a **NamedNodeMap** object.

Object **Notation**

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

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

publicId

This read-only property is of type **String**.

systemId

This read-only property is of type **String**.

Object **Entity**

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

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

publicId

This read-only property is of type **String**.

systemId

This read-only property is of type **String**.

notationName

This read-only property is of type **String**.

Object **EntityReference**

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

Object **ProcessingInstruction**

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

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

target

This read-only property is of type **String**.

data

This property is of type **String**.

D.2: Document Object Model Level 1 HTML

Object **HTMLCollection**

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

length

This read-only property is of type **Number**.

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

item(index)

This method returns a **Node** object.

The **index** parameter is of type **Number**.

Note: This object can also be dereferenced using square bracket notation (e.g. obj[1]). Dereferencing with an integer **index** is equivalent to invoking the **item** method with that index.

namedItem(name)

This method returns a **Node** object.

The **name** parameter is of type **String**.

Note: This object can also be dereferenced using square bracket notation (e.g. obj["foo"]). Dereferencing using a string index is equivalent to invoking the **namedItem** method with that index.

Object **HTMLDocument**

HTMLDocument has the all the properties and methods of the **Document** object 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 read-only property is of type **String**.

domain

This read-only property is of type **String**.

URL

This read-only property is of type **String**.

body

This property is a **HTMLElement** object.

images

This read-only property is a **HTMLCollection** object.

applets

This read-only property is a **HTMLCollection** object.

links

This read-only property is a **HTMLCollection** object.

forms
This read-only property is a **HTMLCollection** object.
anchors
This read-only property is a **HTMLCollection** object.
cookie
This property is of type **String**.

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

open()
This method has no return value.
close()
This method has no return value.
write(text)
This method has no return value.
The **text** parameter is of type **String**.
writeln(text)
This method has no return value.
The **text** parameter is of type **String**.
getElementById(elementId)
This method returns a **Element** object.
The **elementId** parameter is of type **String**.
getElementsByName(elementName)
This method returns a **NodeList** object.
The **elementName** parameter is of type **String**.

Object **HTMLElement**

HTMLElement has the all the properties and methods of the **Element** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

prompt

This property is of type **String**.

Object **HTMLStyleElement**

HTMLStyleElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

elements

This read-only property is a **HTMLCollection** object.

length
This read-only property is a **long** object.
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**.

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

submit()
This method has no return value.
reset()
This method has no return value.

Object **HTMLSelectElement**

HTMLSelectElement has the all the properties and methods of the **HTMLElement** object as well as the properties and methods defined below.

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

type
This read-only property is of type **String**.
selectedIndex
This property is a **long** object.
value
This property is of type **String**.
length
This read-only property is a **long** object.
form
This read-only property is a **HTMLFormElement** object.
options
This read-only property is a **HTMLCollection** object.
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 a **long** object.
tabIndex
This property is a **long** object.

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

add(element, before)

This method has no return value.

The **element** parameter is a **HTMLElement** object.

The **before** parameter is a **HTMLElement** object.

remove(index)

This method has no return value.

The **index** parameter is a **long** object.

blur()

This method has no return value.

focus()

This method has no return value.

Object **HTMLOptGroupElement**

HTMLOptGroupElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

defaultSelected

This property is of type **Boolean**.

text

This read-only property is of type **String**.

index

This read-only property is a **long** object.

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 the **HTMLElement** object 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 read-only property is a **HTMLFormElement** object.
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 a **long** object.
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 a **long** object.
type
This read-only property is of type **String**.
useMap
This property is of type **String**.
value
This property is of type **String**.

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

blur()
This method has no return value.
focus()
This method has no return value.
select()
This method has no return value.
click()
This method has no return value.

Object **HTMLTextAreaElement**

HTMLTextAreaElement has the all the properties and methods of the **HTMLElement** object 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 read-only property is a **HTMLFormElement** object.

accessKey

This property is of type **String**.

cols

This property is a **long** object.

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 a **long** object.

tabIndex

This property is a **long** object.

type

This read-only property is of type **String**.

value

This property is of type **String**.

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

blur()

This method has no return value.

focus()

This method has no return value.

select()

This method has no return value.

Object **HTMLButtonElement**

HTMLButtonElement has the all the properties and methods of the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

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 a **long** object.

type

This read-only property is of type **String**.

value

This property is of type **String**.

Object **HTMLLabelElement**

HTMLLabelElement has the all the properties and methods of the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

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 the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

Object **HTMLLegendElement**

HTMLLegendElement has the all the properties and methods of the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

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 the **HTMLElement** object 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 the **HTMLElement** object 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 a **long** object.

type

This property is of type **String**.

Object **HTMLDListElement**

HTMLDListElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 a **long** object.

Object **HTMLDivElement**

HTMLDivElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

width

This property is a **long** object.

Object **HTMLBRElement**

HTMLBRElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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 a **long** object.
target
This property is of type **String**.
type
This property is of type **String**.

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

blur()
This method has no return value.
focus()
This method has no return value.

Object **HTMLImageElement**

HTMLImageElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

form

This read-only property is a **HTMLFormElement** object.

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 a **long** object.
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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

areas
This read-only property is a **HTMLCollection** object.
name
This property is of type **String**.

Object **HTMLAreaElement**

HTMLAreaElement has the all the properties and methods of the **HTMLElement** object 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 a **long** object.
target
This property is of type **String**.

Object **HTMLScriptElement**

HTMLScriptElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object as well as the properties and methods defined below.

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

caption

This property is a **HTMLTableCaptionElement** object.

tHead

This property is a **HTMLTableSectionElement** object.

tFoot

This property is a **HTMLTableSectionElement** object.

rows

This read-only property is a **HTMLCollection** object.

tBodies

This read-only property is a **HTMLCollection** object.

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**.

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

createTHead()
This method returns a **HTMLElement** object.
deleteTHead()
This method has no return value.
createTFoot()
This method returns a **HTMLElement** object.
deleteTFoot()
This method has no return value.
createCaption()
This method returns a **HTMLElement** object.
deleteCaption()
This method has no return value.
insertRow(index)
This method returns a **HTMLElement** object.
The **index** parameter is a **long** object.
deleteRow(index)
This method has no return value.
The **index** parameter is a **long** object.

Object **HTMLTableCaptionElement**

HTMLTableCaptionElement has the all the properties and methods of the **HTMLElement** object 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 the **HTMLElement** object 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 a **long** object.
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 the **HTMLElement** object 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 read-only property is a **HTMLCollection** object.

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

insertRow(index)

This method returns a **HTMLElement** object.

The **index** parameter is a **long** object.

deleteRow(index)

This method has no return value.

The **index** parameter is a **long** object.

Object **HTMLTableRowElement**

HTMLTableRowElement has the all the properties and methods of the **HTMLElement** object as well as the properties and methods defined below.

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

rowIndex

This read-only property is a **long** object.

sectionRowIndex

This read-only property is a **long** object.

cells

This read-only property is a **HTMLCollection** object.

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** object.

The **index** parameter is a **long** object.

deleteCell(index)

This method has no return value.

The **index** parameter is a **long** object.

Object **HTMLTableCellElement**

HTMLTableCellElement has the all the properties and methods of the **HTMLElement** object as well as the properties and methods defined below.

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

cellIndex

This read-only property is a **long** object.

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 a **long** object.

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 a **long** object.

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 the **HTMLElement** object 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 the **HTMLElement** object 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 the **HTMLElement** object 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**.

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at http://www.w3.org/TR.

G.1: Normative references

Charmod
W3C (World Wide Web Consortium) Character Model for the World Wide Web, November 1999. Available at http://www.w3.org/TR/1999/WD-charmod-19991129

ECMAScript
ECMA (European Computer Manufacturers Association) ECMAScript Language Specification. Available at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

HTML4.0
W3C (World Wide Web Consortium) HTML 4.0 Specification, April 1998. Available at http://www.w3.org/TR/1998/REC-html40-19980424

ISO/IEC 10646
ISO (International Organization for Standardization). ISO/IEC 10646-1:2000 (E). Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization.

Java
Sun Microsystems Inc. The Java Language Specification, James Gosling, Bill Joy, and Guy Steele, September 1996. Available at http://java.sun.com/docs/books/jls

OMGIDL
OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.2. Available at http://www.omg.org/

Unicode
The Unicode Consortium. The Unicode Standard, Version 3.0., February 2000. Available at http://www.unicode.org/unicode/standard/versions/Unicode3.0.html.

XML
W3C (World Wide Web Consortium) Extensible Markup Language (XML) 1.0, February 1998. Available at http://www.w3.org/TR/1998/REC-xml-19980210

G.2: Informative references

COM
Microsoft Corporation The Component Object Model. Available at http://www.microsoft.com/com

CORBA
OMG (Object Management Group) The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available at http://www.omg.org/

HTML3.2
W3C (World Wide Web Consortium) HTML 3.2 Specification, January 1997. Available at http://www.w3.org/TR/REC-html32

Infoset
W3C (World Wide Web Consortium) XML Information Set, December 1999. Available at http://www.w3.org/TR/xml-infoset

JavaIDL
Sun Microsystems Inc. Java IDL. Available at http://java.sun.com/products/jdk/1.2/docs/guide/idl

JavaScript
Netscape Communications Corporation JavaScript Resources. Available at http://developer.netscape.com/one/javascript/resources.html

MIDL
Microsoft Corporation MIDL Language Reference. Available at http://msdn.microsoft.com/library/psdk/midl/mi-laref_1r1h.htm

XHTML10
W3C (World Wide Web Consortium) XHTML 1.0: Extensible HyperText Markup Language, A Reformulation of HTML 4.0 in XML 1.0. Available at http://www.w3.org/TR/2000/REC-xhtml1-20000126

XPointer
W3C (World Wide Web Consortium) XML Pointer Language (XPointer), June 2000. Available at http://www.w3.org/TR/xptr

Objects Index

Document Object Model Core

Document Object Model HTML

Index

Production Notes (Non-Normative)

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.

A. 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.

B. 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/2000/WD-DOM-Level-1-20000929/xml-source.zip

Note: The DOM Level 1 Specification Second Edition has been produced using a DOM Level 2 implementation and an XPath implementation in Java.

C. 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="DOMString" 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.