Document Object Model (DOM) Level 2 Specification

Version 1.0

W3C Candidate Recommendation *10 May, 2000*

This version:
http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
(PostScript file, PDF file, plain text, ZIP file)
Latest version:
http://www.w3.org/TR/DOM-Level-2
Previous version:
http://www.w3.org/TR/2000/CR-DOM-Level-2-20000307

Editors:
Lauren Wood, *SoftQuad Software Inc., chair*
Arnaud Le Hors, *W3C staff contact until October 1999, then IBM*
Vidur Apparao, *Netscape Communications Corporation*
Laurence Cable, *Sun*
Mike Champion, *Arbortext and Software AG*
Mark Davis, *IBM*
Joe Kesselman, *IBM*
Philippe Le Hégaret, *W3C, staff contact (from November 1999)*
Tom Pixley, *Netscape Communications Corporation*
Jonathan Robie, *Texcel Research and Software AG*
Peter Sharpe, *SoftQuad Software Inc.*
Chris Wilson, *Microsoft*

Abstract

This specification defines the Document Object Model Level 2, 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 Level 2 builds on the Document Object Model Level 1.

The DOM Level 2 is made of a set of core interfaces to create and manipulate the structure and contents of a document and a set of optional modules. These modules contain specialized interfaces dedicated to XML, HTML, an abstract view, generic stylesheets, Cascading Style Sheets, Events, traversing the

document structure, and a Range object.

Status of this document

This specification is still in the Candidate Recommendation phase. A coordination issue has arisen, which necessitates an extended Candidate Recommendation phase. It will end when the coordination issue is resolved.

Comments on this document are invited and are to be sent to the public mailing list www-dom@w3.org. An archive is available at http://lists.w3.org/Archives/Public/www-dom/.

Should this specification prove impossible or very difficult to implement, the necessary changes to make it implementable will be made. If this specification is possible to implement, the only changes which will be made to this specification are minor editorial changes and clarifications.

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.

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

Note: The coordination issue affects the handling of namespace URIs. The resolution of the coordination issue may necessitate changes to the DOM Level 2 Core module.

Table of contents

Table of contents

Expanded Table of Contents

Copyright Notice

W3C Document Copyright Notice and License

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

What is the Document Object Model?

Editors
Jonathan Robie, Software AG

Introduction

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

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

As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used with any programming language. In order to provide a precise, language-independent specification of the DOM interfaces, we have chosen to define the specifications in Object Management Group (OMG) IDL [OMGIDL], as defined in the CORBA 2.3.1 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 [JavaScript] and JScript [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 ([COM], [JavaIDL], [MIDL], ...). 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.

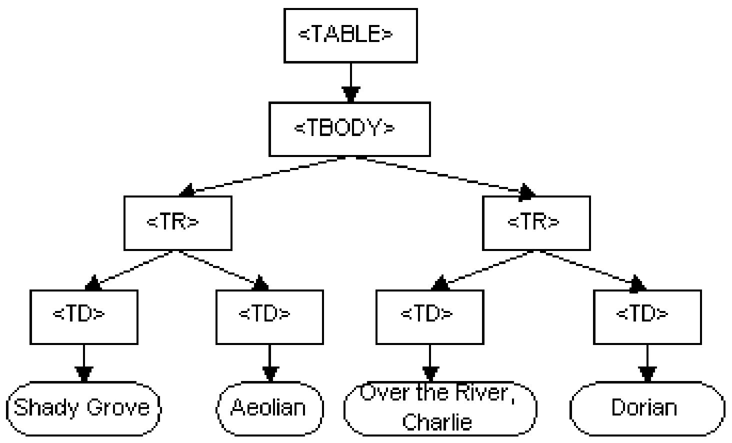
What the Document Object Model is

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.

- 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 "&amp;" 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 "&amp;" 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 DOM Level 1 [DOM-Level-1].

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 2 consists of several modules: Core, HTML, Views, StyleSheets, CSS, Events, Traversal, and Range. 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. The other modules are optional.

A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether the module is supported or not. The feature strings for all modules in DOM Level 2 are listed in the following table; (strings are case-insensitive):

Module	Feature String
XML	XML
HTML	HTML
Views	Views
StyleSheets	StyleSheets
CSS	CSS
CSS (extended interfaces)	CSS2
Events	Events
User Interface Events (`UIEvent` [p.231] interface)	UIEvents
Mouse Events (`MouseEvents` [p.233] interface)	MouseEvents
Mutation Events (`MutationEvent` [p.238] interface)	MutationEvents
HTML Events	HTMLEvents
Traversal	Traversal
Range	Range

The following table contains all dependencies between modules:

Module	Implies
Views	XML or HTML
StyleSheets	StyleSheets and XML or HTML
CSS	StyleSheets, Views and XML or HTML
CSS2	CSS, StyleSheets, Views and XML or HTML
Events	XML or HTML
UIEvents	Views, Events and XML or HTML
MouseEvents	UIEvents, Views, Events and XML or HTML
MutationEvents	Events and XML or HTML
HTMLEvents	Events and HTML
Traversal	XML or HTML
Range	XML or 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.

The Level 1 interfaces were extended to provide both Level 1 and Level 2 functionality.

DOM implementations in languages other than Java or ECMA Script may choose bindings that are appropriate and natural for their language and run time environment. For example, some systems may need to create a Document2 class which inherits from Document and contains the new methods and attributes.

1. Document Object Model Core

Editors
Arnaud Le Hors, W3C
Mike Champion, ArborText (for DOM Level 1 from November 20, 1997)
Steve Byrne, JavaSoft (for DOM Level 1 until November 19, 1997)
Gavin Nicol, Inso EPS (for DOM Level 1)
Lauren Wood, SoftQuad, Inc. (for DOM Level 1)

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.29] 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.38] 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.29] -- `Element` [p.57] (maximum of one), `ProcessingInstruction` [p.71] , `Comment` [p.66] , `DocumentType` [p.68] (maximum of one)
- `DocumentFragment` [p.28] -- `Element` [p.57] , `ProcessingInstruction` [p.71] , `Comment` [p.66] , `Text` [p.66] , `CDATASection` [p.67] , `EntityReference` [p.70]
- `DocumentType` [p.68] -- no children
- `EntityReference` [p.70] -- `Element` [p.57] , `ProcessingInstruction` [p.71] , `Comment` [p.66] , `Text` [p.66] , `CDATASection` [p.67] , `EntityReference`
- `Element` [p.57] -- `Element`, `Text` [p.66] , `Comment` [p.66] , `ProcessingInstruction` [p.71] , `CDATASection` [p.67] , `EntityReference` [p.70]
- `Attr` [p.56] -- `Text` [p.66] , `EntityReference` [p.70]
- `ProcessingInstruction` [p.71] -- no children
- `Comment` [p.66] -- no children
- `Text` [p.66] -- no children
- `CDATASection` [p.67] -- no children
- `Entity` [p.69] -- `Element` [p.57] , `ProcessingInstruction` [p.71] , `Comment` [p.66] , `Text` [p.66] , `CDATASection` [p.67] , `EntityReference` [p.70]
- `Notation` [p.69] -- no children

The DOM also specifies a `NodeList` [p.47] interface to handle ordered lists of `Nodes` [p.38] , such as the children of a `Node` [p.38] , or the elements returned by the `getElementsByTagName` method of the `Element` [p.57] interface, and also a `NamedNodeMap` [p.48] interface to handle unordered sets of nodes referenced by their name attribute, such as the attributes of an `Element`. `NodeList` [p.47] and `NamedNodeMap` [p.48] 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.57] , 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.38] in the tree are reflected in all references to that `Node` in `NodeList` and `NamedNodeMap` objects.

Finally, the interfaces `Text` [p.66] , `Comment` [p.66] , and `CDATASection` [p.67] all inherit from the `CharacterData` [p.52] 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. Objects implementing some interface "X" are created by a "createX()" method on the `Document` [p.29] interface; this is because all DOM objects live in the context of a specific Document.

The DOM Level 2 API does *not* define a standard way to create `DOMImplementation` [p.26] 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.38] 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.38] 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.57] 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.21] is a sequence of *16-bit units* [p.447] .
  IDL Definition

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

- Applications must encode `DOMString` [p.21] 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.21] (a high surrogate and a low surrogate).

  Note: Even though the DOM defines the name of the string type to be `DOMString` [p.21] , 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. The `DOMTimeStamp` type

To ensure interoperability, the DOM specifies the following:

- **Type Definition *DOMTimeStamp***

  A `DOMTimeStamp` [p.22] represents a number of milliseconds.
  IDL Definition

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

- **Note:** Even though the DOM uses the type `DOMTimeStamp` [p.22] , bindings may use different types. For example for Java, `DOMTimeStamp` is bound to the `long` type. In ECMAScript, `TimeStamp` is bound to the `Date` type because the range of the `integer` type is too small.

1.1.7. 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.447] of the `DOMString` [p.21] . 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].

1.1.8. XML Namespaces

The DOM Level 2 supports XML namespaces [Namespaces] by augmenting several interfaces of the DOM Level 1 Core to allow creating and manipulating elements and attributes associated to a namespace.

As far as the DOM is concerned, special attributes used for declaring XML namespaces are still exposed and can be manipulated just like any other attribute. However, nodes are permanently bound to *namespace URIs* [p.450] as they get created. Consequently, moving a node within a document, using the DOM, in no case results in a change of its *namespace prefix* [p.450] or namespace URI. Similarly, creating a node with a namespace prefix and namespace URI, or changing the namespace prefix of a node, does not result in any addition, removal, or modification of any special attributes for declaring the appropriate XML namespaces. Namespace validation is not enforced; the DOM application is responsible. In particular, since the mapping between prefixes and namespace URIs is not enforced, in general, the resulting document cannot be serialized naively. For example, applications may have to declare every namespace in use when serializing a document.

Namespace URIs are treated literally, whether they are absolute or relative. No processing of the namespace URI such as "absolutization" or "canonicalization" is performed by the DOM implementation.

Note: In the DOM, all namespace declaration attributes are *by definition* bound to the namespace URI: "http://www.w3.org/2000/xmlns/". These are the attributes whose *namespace prefix* [p.450] or *qualified name* [p.450] is "xmlns". Although, at the time of writing, this is not part of the XML Namespaces specification[Namespaces], it is planned to be incorporated in a future revision.

In a document with no namespaces, the child list of an `EntityReference` [p.70] node is always the same as that of the corresponding `Entity` [p.69] . This is not true in a document where an entity contains unbound *namespace prefixes* [p.450] . In such a case, the descendants of the corresponding `EntityReference` nodes may be bound to different *namespace URIs* [p.450] , depending on where the entity references are. Also, because, in the DOM, nodes always remain bound to the same namespace URI, moving such `EntityReference` nodes can lead to documents that cannot be serialized. This is also true when the DOM Level 1 method `createEntityReference` of the `Document` [p.29] interface is used to create entity references that correspond to such entities, since the descendants of the returned `EntityReference` are unbound. The DOM Level 2 does not support any mechanism to resolve namespace prefixes. For all of these reasons, use of such entities and entity references should be avoided or used with extreme care. A future Level of the DOM may include some additional support for handling these.

The new methods, such as `createElementNS` and `createAttributeNS` of the `Document` [p.29] interface, are meant to be used by namespace aware applications. Simple applications that do not use namespaces can use the DOM Level 1 methods, such as `createElement` and `createAttribute`. Elements and attributes created in this way do not have any namespace prefix, namespace URI, or local name.

Note: DOM Level 1 methods are namespace ignorant. Therefore, while it is safe to use these methods when not dealing with namespaces, using them and the new ones at the same time should be avoided. DOM Level 1 methods solely identify attribute nodes by their `nodeName`. On the contrary, the DOM Level 2 methods related to namespaces, identify attribute nodes by their `namespaceURI` and `localName`. Because of this fundamental difference, mixing both sets of methods can lead to unpredictable results. In particular, using `setAttributeNS`, an element may have two attributes (or more) that have the same `nodeName`, but different `namespaceURI`s. Calling `getAttribute` with that `nodeName` could then return any of those attributes. The result depends on the implementation. Similarly, using `setAttributeNode`, one can set two attributes (or more) that have different `nodeNames` but the same `prefix` and `namespaceURI`. In this case `getAttributeNodeNS` will return either attribute, in an implementation dependent manner. The only guarantee in such cases is that all methods that access a named item by its `nodeName` will access the same item, and all methods which access a node by its URI and local name will access the same node. For instance, `setAttribute` and `setAttributeNS` affect the node that `getAttribute` and `getAttributeNS`, respectively, return.

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.47] .

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;
// Introduced in DOM Level 2:
const unsigned short      INVALID_STATE_ERR              = 11;
```

```
    // Introduced in DOM Level 2:
    const unsigned short      SYNTAX_ERR                     = 12;
    // Introduced in DOM Level 2:
    const unsigned short      INVALID_MODIFICATION_ERR       = 13;
    // Introduced in DOM Level 2:
    const unsigned short      NAMESPACE_ERR                  = 14;
    // Introduced in DOM Level 2:
    const unsigned short      INVALID_ACCESS_ERR             = 15;
```

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_ACCESS_ERR` , introduced in **DOM Level 2**.
If a parameter or an operation is not supported by the underlying object.

`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.

`INVALID_MODIFICATION_ERR` , introduced in **DOM Level 2**.
If an attempt is made to modify the type of the underlying object.

`INVALID_STATE_ERR` , introduced in **DOM Level 2**.
If an attempt is made to use an object that is not, or is no longer, usable.

`NAMESPACE_ERR` , introduced in **DOM Level 2**.
If an attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

`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

`SYNTAX_ERR` , introduced in **DOM Level 2**.
: If an invalid or illegal string is specified.

`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.

IDL Definition

```
interface DOMImplementation {
  boolean            hasFeature(in DOMString feature,
                                in DOMString version);
  // Introduced in DOM Level 2:
  DocumentType       createDocumentType(in DOMString qualifiedName,
                                        in DOMString publicId,
                                        in DOMString systemId)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  Document           createDocument(in DOMString namespaceURI,
                                    in DOMString qualifiedName,
                                    in DocumentType doctype)
                                        raises(DOMException);
};
```

Methods

`createDocument` introduced in **DOM Level 2**
: Creates an XML `Document` [p.29] object of the specified type with its document element. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]
: The *namespace URI* [p.450] of the document element to create.

`qualifiedName` of type `DOMString`
: The *qualified name* [p.450] of the document element to be created.

`doctype` of type `DocumentType` [p.68]
: The type of document to be created or `null`.
When `doctype` is not `null`, its `Node.ownerDocument` attribute is set to the

document being created.

Return Value

`Document` [p.29]	A new `Document` object.

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character. NAMESPACE_ERR: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is `null` or an empty string, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace" [Namespaces]. WRONG_DOCUMENT_ERR: Raised if `doctype` has already been used with a different document or was created from a different implementation.

`createDocumentType` introduced in **DOM Level 2**

Creates an empty `DocumentType` [p.68] node. Entity declarations and notations are not made available. Entity reference expansions and default attribute additions do not occur. It is expected that a future version of the DOM will provide a way for populating a `DocumentType`.
HTML-only DOM implementations do not need to implement this method.

Parameters

`qualifiedName` of type `DOMString` [p.21]
The *qualified name* [p.450] of the document type to be created.

`publicId` of type `DOMString`
The external subset public identifier.

`systemId` of type `DOMString`
The external subset system identifier.

Return Value

`DocumentType` [p.68]	A new `DocumentType` node with `Node.ownerDocument` set to `null`.

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character. NAMESPACE_ERR: Raised if the `qualifiedName` is malformed.

`hasFeature`

Test if the DOM implementation implements a specific feature.

Parameters

`feature` of type `DOMString` [p.21]

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.16] section. The name must be an *XML name* [p.451] . 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 2, this is the string "2.0". 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.29] 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.38] -- 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.66] 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.29] (or indeed any other `Node` [p.38] 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.38] 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);
  // Introduced in DOM Level 2:
  Node               importNode(in Node importedNode,
                                in boolean deep)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  Element            createElementNS(in DOMString namespaceURI, 
                                     in DOMString qualifiedName)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  Attr               createAttributeNS(in DOMString namespaceURI, 
                                       in DOMString qualifiedName)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList           getElementsByTagNameNS(in DOMString namespaceURI, 
                                            in DOMString localName);
  // Introduced in DOM Level 2:
  Element            getElementById(in DOMString elementId);
};
```

Attributes

`doctype` of type `DocumentType` [p.68] , readonly

The Document Type Declaration (see `DocumentType` [p.68]) associated with this document. For HTML documents as well as XML documents without a document type declaration this returns `null`. The DOM Level 2 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.38] interface.

`documentElement` of type `Element` [p.57] , 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.26] , readonly

The `DOMImplementation` [p.26] object that handles this document. A DOM application may use objects from multiple implementations.

Methods

`createAttribute`

Creates an `Attr` [p.56] of the given name. Note that the `Attr` instance can then be set on an `Element` [p.57] using the `setAttributeNode` method.
To create an attribute with a qualified name and namespace URI, use the `createAttributeNS` method.

Parameters

`name` of type `DOMString` [p.21]

The name of the attribute.

Return Value

`Attr` [p.56]	A new `Attr` object with the `nodeName` attribute set to `name`, and `localName`, `prefix`, and `namespaceURI` set to `null`.

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createAttributeNS` introduced in **DOM Level 2**

Creates an attribute of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]

The *namespace URI* [p.450] of the attribute to create.

`qualifiedName` of type `DOMString`

The *qualified name* [p.450] of the attribute to instantiate.

Return Value

`Attr` [p.56]

A new `Attr` object with the following attributes:

Attribute	Value
`Node.nodeName`	qualifiedName
`Node.namespaceURI`	`namespaceURI`
`Node.prefix`	prefix, extracted from `qualifiedName`, or `null` if there is no prefix
`Node.localName`	*local name* [p.450] , extracted from `qualifiedName`
`Attr.name`	`qualifiedName`

Exceptions

`DOMException` [p.24]

INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.

NAMESPACE_ERR: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is `null` or an empty string, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace", or if the `qualifiedName` is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/".

`createCDATASection`

Creates a `CDATASection` [p.67] node whose value is the specified string.

Parameters

`data` of type `DOMString` [p.21]

The data for the `CDATASection` [p.67] contents.

Return Value

`CDATASection` [p.67]	The new `CDATASection` object.

Exceptions

`DOMException` [p.24]	NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`createComment`

Creates a `Comment` [p.66] node given the specified string.

Parameters

`data` of type `DOMString` [p.21]

The data for the node.

Return Value

`Comment` [p.66]	The new `Comment` object.

No Exceptions

`createDocumentFragment`

Creates an empty `DocumentFragment` [p.28] object.

Return Value

`DocumentFragment` [p.28]	A new `DocumentFragment`.

No Parameters

No Exceptions

`createElement`

Creates an element of the type specified. Note that the instance returned implements the `Element` [p.57] interface, so attributes can be specified directly on the returned object. In addition, if there are known attributes with default values, `Attr` [p.56] nodes representing them are automatically created and attached to the element.

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

Parameters

`tagName` of type `DOMString` [p.21]
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.57]	A new `Element` object with the `nodeName` attribute set to `tagName`, and `localName`, `prefix`, and `namespaceURI` set to `null`.

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified name contains an illegal character.

`createElementNS` introduced in **DOM Level 2**
Creates an element of the given qualified name and namespace URI. HTML-only DOM implementations do not need to implement this method.
Parameters
`namespaceURI` of type `DOMString` [p.21]
The *namespace URI* [p.450] of the element to create.

`qualifiedName` of type `DOMString`
The *qualified name* [p.450] of the element type to instantiate.

Return Value

`Element` [p.57] A new `Element` object with the following attributes:

Attribute	Value
`Node.nodeName`	`qualifiedName`
`Node.namespaceURI`	`namespaceURI`
`Node.prefix`	prefix, extracted from `qualifiedName`, or `null` if there is no prefix
`Node.localName`	*local name* [p.450] , extracted from `qualifiedName`
`Element.tagName`	`qualifiedName`

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character. NAMESPACE_ERR: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is `null` or an empty string, or if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace" [Namespaces].

`createEntityReference`

Creates an `EntityReference` [p.70] 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.69] node.

Note: If any descendant of the `Entity` [p.69] node has an unbound *namespace prefix* [p.450] , the corresponding descendant of the created `EntityReference` [p.70] node is also unbound; (its `namespaceURI` is `null`). The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

Parameters

`name` of type `DOMString` [p.21]

The name of the entity to reference.

Return Value

`EntityReference` [p.70]	The new `EntityReference` object.

Exceptions

`DOMException` [p.24]	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.71] node given the specified name and data strings.

Parameters

`target` of type `DOMString` [p.21]

The target part of the processing instruction.

`data` of type `DOMString`
The data for the node.

Return Value

`ProcessingInstruction` [p.71]	The new `ProcessingInstruction` object.

Exceptions

`DOMException` [p.24]	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.66] node given the specified string.
Parameters
`data` of type `DOMString` [p.21]
The data for the node.

Return Value

`Text` [p.66]	The new `Text` object.

No Exceptions

`getElementById` introduced in **DOM Level 2**
Returns the `Element` [p.57] 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`.

Note: The DOM implementation must have information that says which attributes are of type ID. Attributes with the name "ID" are not of type ID unless so defined. Implementations that do not know whether attributes are of type ID or not are expected to return `null`.

Parameters
`elementId` of type `DOMString` [p.21]
The unique `id` value for an element.

Return Value

`Element` [p.57]	The matching element.

No Exceptions

`getElementsByTagName`

Returns a `NodeList` [p.47] of all the `Elements` [p.57] 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.21]

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

Return Value

`NodeList` [p.47]	A new `NodeList` object containing all the matched `Elements` [p.57] .

No Exceptions

`getElementsByTagNameNS` introduced in **DOM Level 2**

Returns a `NodeList` [p.47] of all the `Elements` [p.57] with a given *local name* [p.450] and namespace URI in the order in which they are encountered in a preorder traversal of the `Document` tree.

Parameters

`namespaceURI` of type `DOMString` [p.21]

The *namespace URI* [p.450] of the elements to match on. The special value "*" matches all namespaces.

`localName` of type `DOMString`

The *local name* [p.450] of the elements to match on. The special value "*" matches all local names.

Return Value

`NodeList` [p.47]	A new `NodeList` object containing all the matched `Elements` [p.57] .

No Exceptions

`importNode` introduced in **DOM Level 2**

Imports a node from another document to this document. The returned node has no parent; (`parentNode` is `null`). The source node is not altered or removed from the original document; this method creates a new copy of the source node.

For all nodes, importing a node creates a node object owned by the importing document, with attribute values identical to the source node's `nodeName` and `nodeType`, plus the attributes related to namespaces (`prefix`, `localName`, and `namespaceURI`). As in the `cloneNode` operation on a `Node` [p.38] , the source node is not altered.

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

ATTRIBUTE_NODE
The `ownerElement` attribute is set to `null` and the `specified` flag is set to `true` on the generated `Attr` [p.56] . The descendants of the source `Attr` are recursively imported and the resulting nodes reassembled to form the corresponding subtree.
Note that the `deep` parameter has no effect on `Attr` [p.56] nodes; they always carry their children with them when imported.

DOCUMENT_FRAGMENT_NODE
If the `deep` option was set to `true`, the descendants of the source element will be recursively imported and the resulting nodes reassembled to form the corresponding subtree. Otherwise, this simply generates an empty `DocumentFragment` [p.28] .

DOCUMENT_NODE
`Document` nodes cannot be imported.

DOCUMENT_TYPE_NODE
`DocumentType` [p.68] nodes cannot be imported.

ELEMENT_NODE
Specified attribute nodes of the source element are imported, and the generated `Attr` [p.56] nodes are attached to the generated `Element` [p.57] . Default attributes are *not* copied, though if the document being imported into defines default attributes for this element name, those are assigned. If the `importNode` `deep` parameter was set to `true`, the descendants of the source element will be recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_NODE
`Entity` [p.69] nodes can be imported, however in the current release of the DOM the `DocumentType` [p.68] is readonly. Ability to add these imported nodes to a `DocumentType` will be considered for addition to a future release of the DOM.
On import, the `publicId`, `systemId`, and `notationName` attributes are copied.
If a `deep` import is requested, the descendants of the the source `Entity` [p.69] is recursively imported and the resulting nodes reassembled to form the corresponding subtree.

ENTITY_REFERENCE_NODE
Only the `EntityReference` [p.70] itself is copied, even if a `deep` import is requested, since the source and destination documents might have defined the entity differently. If the document being imported into provides a definition for this entity name, its value is assigned.

NOTATION_NODE
`Notation` [p.69] nodes can be imported, however in the current release of the DOM the `DocumentType` [p.68] is readonly. Ability to add these imported nodes to a `DocumentType` will be considered for addition to a future release of the DOM.
On import, the `publicId` and `systemId` attributes are copied.
Note that the `deep` parameter has no effect on `Notation` [p.69] nodes since they never have any children.

PROCESSING_INSTRUCTION_NODE
The imported node copies its `target` and `data` values from those of the source

node.

TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE

These three types of nodes inheriting from `CharacterData` [p.52] copy their `data` and `length` attributes from those of the source node.

Parameters

`importedNode` of type `Node` [p.38]

The node to import.

`deep` of type `boolean`

If `true`, recursively import the subtree under the specified node; if `false`, import only the node itself, as explained above. This has no effect on `Attr` [p.56] , `EntityReference` [p.70] , and `Notation` [p.69] nodes.

Return Value

`Node` [p.38]	The imported node that belongs to this `Document`.

Exceptions

`DOMException` [p.24]	NOT_SUPPORTED_ERR: Raised if the type of node being imported is not supported.

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.66] nodes may not have children, and adding children to such nodes results in a `DOMException` [p.24] 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` [p.57] or `attributes` for a `Comment` [p.66]), 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;
  // Modified in DOM Level 2:
  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);
  // Introduced in DOM Level 2:
  void               normalize();
  // Introduced in DOM Level 2:
  boolean            supports(in DOMString feature,
                              in DOMString version);
  // Introduced in DOM Level 2:
  readonly attribute DOMString        namespaceURI;
  // Introduced in DOM Level 2:
           attribute DOMString        prefix;
                                        // raises(DOMException) on setting

  // Introduced in DOM Level 2:
  readonly attribute DOMString        localName;
};
```

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.56] .

`CDATA_SECTION_NODE`
: The node is a `CDATASection` [p.67] .

`COMMENT_NODE`
: The node is a `Comment` [p.66] .

`DOCUMENT_FRAGMENT_NODE`
: The node is a `DocumentFragment` [p.28] .

`DOCUMENT_NODE`
: The node is a `Document` [p.29] .

`DOCUMENT_TYPE_NODE`
: The node is a `DocumentType` [p.68] .

`ELEMENT_NODE`
: The node is an `Element` [p.57] .

`ENTITY_NODE`
: The node is an `Entity` [p.69] .

`ENTITY_REFERENCE_NODE`
: The node is an `EntityReference` [p.70] .

`NOTATION_NODE`
: The node is a `Notation` [p.69] .

`PROCESSING_INSTRUCTION_NODE`
: The node is a `ProcessingInstruction` [p.71] .

`TEXT_NODE`
: The node is a `Text` [p.66] 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.48] , readonly
A `NamedNodeMap` [p.48] containing the attributes of this node (if it is an `Element` [p.57]) or `null` otherwise.

`childNodes` of type `NodeList` [p.47] , readonly
A `NodeList` [p.47] that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes.

`firstChild` of type `Node` [p.38] , readonly
The first child of this node. If there is no such node, this returns `null`.

`lastChild` of type `Node` [p.38] , readonly
The last child of this node. If there is no such node, this returns `null`.

`localName` of type `DOMString` [p.21] , readonly, introduced in **DOM Level 2**
Returns the local part of the *qualified name* [p.450] of this node.
For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` [p.29] interface, this is always `null`.

`namespaceURI` of type `DOMString` [p.21] , readonly, introduced in **DOM Level 2**

The *namespace URI* [p.450] of this node, or `null` if it is unspecified.
This is not a computed value that is the result of a namespace lookup based on an examination of the namespace declarations in scope. It is merely the namespace URI given at creation time.
For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` [p.29] interface, this is always `null`.

Note: Per the *Namespaces in XML* Specification [Namespaces] an attribute does not inherit its namespace from the element it is attached to. If an attribute is not explicitly given a namespace, it simply has no namespace.

`nextSibling` of type `Node` [p.38] , readonly

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

`nodeName` of type `DOMString` [p.21] , 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.21]

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.24]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Exceptions on retrieval

`DOMException` [p.24]	DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` [p.21] variable on the implementation platform.

`ownerDocument` of type `Document` [p.29] , readonly, modified in **DOM Level 2**

The `Document` [p.29] object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document` or a `DocumentType` [p.68] which is not used with any `Document` yet, this is `null`.

`parentNode` of type `Node` [p.38] , readonly

The parent of this node. All nodes, except `Attr` [p.56] , `Document` [p.29] , `DocumentFragment` [p.28] , `Entity` [p.69] , and `Notation` [p.69] 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`.

`prefix` of type `DOMString` [p.21] , introduced in **DOM Level 2**

The *namespace prefix* [p.450] of this node, or `null` if it is unspecified.
Note that setting this attribute, when permitted, changes the `nodeName` attribute, which holds the *qualified name* [p.450] , as well as the `tagName` and `name` attributes of the `Element` [p.57] and `Attr` [p.56] interfaces, when applicable.
Note also that changing the prefix of an attribute that is known to have a default value, does not make a new attribute with the default value and the original prefix appear, since the `namespaceURI` and `localName` do not change.
For nodes of any type other than `ELEMENT_NODE` and `ATTRIBUTE_NODE` and nodes created with a DOM Level 1 method, such as `createElement` from the `Document` [p.29] interface, this is always `null`.

Exceptions on setting

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified prefix contains an illegal character. NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly. NAMESPACE_ERR: Raised if the specified `prefix` is malformed, if the `namespaceURI` of this node is `null`, if the specified prefix is "xml" and the `namespaceURI` of this node is different from "http://www.w3.org/XML/1998/namespace", if this node is an attribute and the specified prefix is "xmlns" and the `namespaceURI` of this node is different from "http://www.w3.org/2000/xmlns/", or if this node is an attribute and the `qualifiedName` of this node is "xmlns" [Namespaces].

`previousSibling` of type `Node` [p.38] , 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.38]

The node to add.
If it is a `DocumentFragment` [p.28] object, the entire contents of the document fragment are moved into the child list of this node

Return Value

`Node` [p.38]	The node added.

Exceptions

`DOMException` [p.24]	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.57] 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.66] 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.70] clone are *readonly* [p.450] . In addition, clones of unspecified `Attr` [p.56] nodes are specified. And, cloning `Document` [p.29] , `DocumentType` [p.68] , `Entity` [p.69] , and `Notation` [p.69] 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.57]).

Return Value

`Node` [p.38]	The duplicate node.

No Exceptions

`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.28] 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.38]
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.38]	The node being inserted.

Exceptions

`DOMException` [p.24]	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.

`normalize` introduced in **DOM Level 2**

Puts all `Text` [p.66] nodes in the full depth of the sub-tree underneath this `Node`, including attribute nodes, into a "normal" form where only structure (e.g., elements, comments, processing instructions, CDATA sections, and entity references) separates `Text` nodes, i.e., there are neither adjacent `Text` nodes nor empty `Text` nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as XPointer lookups) that depend on a particular document tree structure are to be used.

Note: In cases where the document contains `CDATASections` [p.67] , the normalize operation alone may not be sufficient, since XPointers do not differentiate between `Text` [p.66] nodes and `CDATASection` [p.67] nodes.

No Parameters
No Return Value
No Exceptions

`removeChild`

Removes the child node indicated by `oldChild` from the list of children, and returns it.
Parameters
`oldChild` of type `Node` [p.38]
The node being removed.

Return Value

`Node` [p.38]	The node removed.

Exceptions

`DOMException` [p.24]	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.28] 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.38]
The new node to put in the child list.

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

Return Value

`Node` [p.38]	The node replaced.

Exceptions

`DOMException` [p.24]	HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to put in is one of this node's ancestors. 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.

`supports` introduced in **DOM Level 2**

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.

Parameters

`feature` of type `DOMString` [p.21]

The name of the feature to test. This is the same name which can be passed to the method `hasFeature` on `DOMImplementation` [p.26] .

`version` of type `DOMString`

This is the version number of the feature to test. In Level 2, version 1, this is the string "2.0". If the version is not specified, supporting any version of the feature will cause the method to return `true`.

Return Value

`boolean`	Returns `true` if the specified feature is supported on this node, `false` otherwise.

No Exceptions

Interface *NodeList*

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented. `NodeList` objects in the DOM are *live* [p.20] .

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.38]	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.47] ; `NamedNodeMaps` are not maintained in any particular order. Objects contained in an object implementing `NamedNodeMap` may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a `NamedNodeMap`, and does not imply that the DOM specifies an order to these Nodes.

`NamedNodeMap` objects in the DOM are *live* [p.20] .

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;
  // Introduced in DOM Level 2:
  Node               getNamedItemNS(in DOMString namespaceURI,
                                    in DOMString localName);
  // Introduced in DOM Level 2:
  Node               setNamedItemNS(in Node arg)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
```

```
  Node               removeNamedItemNS(in DOMString namespaceURI,
                                       in DOMString localName)
                                        raises(DOMException);
};
```

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.21]

The `nodeName` of a node to retrieve.

Return Value

`Node` [p.38]	A `Node` (of any type) with the specified `nodeName`, or `null` if it does not identify any node in this map.

No Exceptions

`getNamedItemNS` introduced in **DOM Level 2**

Retrieves a node specified by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]

The *namespace URI* [p.450] of the node to retrieve.

`localName` of type `DOMString`

The *local name* [p.450] of the node to retrieve.

Return Value

`Node` [p.38]	A `Node` (of any type) with the specified local name and namespace URI, or `null` if they do not identify any node in this map.

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.38]	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. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the attributes attribute of the `Node` [p.38] interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.
Parameters
`name` of type `DOMString` [p.21]
The `nodeName` of the node to remove.

Return Value

`Node` [p.38]	The node removed from this map if a node with such a name exists.

Exceptions

`DOMException` [p.24]	NOT_FOUND_ERR: Raised if there is no node named `name` in this map.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`removeNamedItemNS` introduced in **DOM Level 2**
Removes a node specified by local name and namespace URI. A removed attribute may be known to have a default value when this map contains the attributes attached to an element, as returned by the attributes attribute of the `Node` [p.38] interface. If so, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.
HTML-only DOM implementations do not need to implement this method.
Parameters
`namespaceURI` of type `DOMString` [p.21]
The *namespace URI* [p.450] of the node to remove.

`localName` of type `DOMString`
The *local name* [p.450] of the node to remove.

Return Value

`Node` [p.38]	The node removed from this map if a node with such a local name and namespace URI exists.

Exceptions

`DOMException` [p.24]	NOT_FOUND_ERR: Raised if there is no node with the specified `namespaceURI` and `localName` in this map. NO_MODIFICATION_ALLOWED_ERR: Raised if this map is readonly.

`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.38]
A node to store in this map. The node will later be accessible using the value of its `nodeName` attribute.

Return Value

`Node` [p.38]	If the new `Node` replaces an existing node the replaced `Node` is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.24]	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.56] that is already an attribute of another `Element` [p.57] object. The DOM user must explicitly clone `Attr` nodes to re-use them in other elements.

`setNamedItemNS` introduced in **DOM Level 2**

Adds a node using its `namespaceURI` and `localName`. If a node with that namespace URI and that local name is already present in this map, it is replaced by the new one. HTML-only DOM implementations do not need to implement this method.

Parameters

`arg` of type `Node` [p.38]

A node to store in this map. The node will later be accessible using the value of its `namespaceURI` and `localName` attributes.

Return Value

`Node` [p.38]	If the new `Node` replaces an existing node the replaced `Node` is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.24]	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.56] that is already an attribute of another `Element` [p.57] 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.66] and others do inherit the interface from it. All `offsets` in this interface start from 0.

As explained in the `DOMString` [p.21] 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.447] 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.21]

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.21] . In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

Exceptions on setting

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Exceptions on retrieval

`DOMException` [p.24]	DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` [p.21] variable on the implementation platform.

`length` of type `unsigned long`, readonly

The number of *16-bit units* [p.447] 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.21] specified.

Parameters

`arg` of type `DOMString` [p.21]

The `DOMString` to append.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`deleteData`

Remove a range of *16-bit units* [p.447] 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.24]	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 character offset.

Parameters

`offset` of type `unsigned long`

The character offset at which to insert.

`arg` of type `DOMString` [p.21]

The `DOMString` to insert.

Exceptions

`DOMException` [p.24]	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.447] 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.21]

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

Exceptions

`DOMException` [p.24]	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.21]	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.24]	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.21] .

Interface *Attr*

The `Attr` interface represents an attribute in an `Element` [p.57] object. Typically the allowable values for the attribute are defined in a document type definition.

`Attr` objects inherit the `Node` [p.38] 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.28] . However, they can be associated with `Element` [p.57] 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.66] or `EntityReference` [p.70] 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;
           attribute DOMString        value;
                                        // raises(DOMException) on setting

  // Introduced in DOM Level 2:
  readonly attribute Element          ownerElement;
};
```

Attributes

`name` of type `DOMString` [p.21] , readonly
Returns the name of this attribute.

`ownerElement` of type `Element` [p.57] , readonly, introduced in **DOM Level 2**
The `Element` [p.57] node this attribute is attached to or `null` if this attribute is not in use.

`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 `ownerElement` attribute is `null` (i.e. because it was just created or was set to `null` by the various removal and cloning operations) `specified` is `true`.

`value` of type `DOMString` [p.21]
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.57] interface.
On setting, this creates a `Text` [p.66] 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.57] interface.
Exceptions on setting

`DOMException` [p.24]	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.38] , 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.56] object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an `Attr` object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience.

Note: In DOM Level 2, the method `normalize` is inherited from the `Node` [p.38] interface where it was moved.

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);
  // Introduced in DOM Level 2:
  DOMString          getAttributeNS(in DOMString namespaceURI, 
                                    in DOMString localName);
  // Introduced in DOM Level 2:
  void               setAttributeNS(in DOMString namespaceURI, 
                                    in DOMString qualifiedName, 
                                    in DOMString value)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  void               removeAttributeNS(in DOMString namespaceURI, 
                                       in DOMString localName)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  Attr               getAttributeNodeNS(in DOMString namespaceURI, 
                                        in DOMString localName);
  // Introduced in DOM Level 2:
  Attr               setAttributeNodeNS(in Attr newAttr)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList           getElementsByTagNameNS(in DOMString namespaceURI, 
                                            in DOMString localName);
  // Introduced in DOM Level 2:
  boolean            hasAttribute(in DOMString name);
  // Introduced in DOM Level 2:
  boolean            hasAttributeNS(in DOMString namespaceURI, 
                                    in DOMString localName);
};
```

Attributes

`tagName` of type `DOMString` [p.21] , 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.21]

The name of the attribute to retrieve.

Return Value

`DOMString` [p.21]	The `Attr` [p.56] value as a string, or the empty string if that attribute does not have a specified or default value.

No Exceptions

`getAttributeNS` introduced in **DOM Level 2**

Retrieves an attribute value by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]

The *namespace URI* [p.450] of the attribute to retrieve.

`localName` of type `DOMString`

The *local name* [p.450] of the attribute to retrieve.

Return Value

`DOMString` [p.21]	The `Attr` [p.56] value as a string, or the empty string if that attribute does not have a specified or default value.

No Exceptions

`getAttributeNode`

Retrieves an attribute node by name.

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

Parameters

`name` of type `DOMString` [p.21]
The name (`nodeName`) of the attribute to retrieve.

Return Value

`Attr` [p.56]	The `Attr` node with the specified name (`nodeName`) or `null` if there is no such attribute.

No Exceptions

`getAttributeNodeNS` introduced in **DOM Level 2**
Retrieves an `Attr` [p.56] node by local name and namespace URI. HTML-only DOM implementations do not need to implement this method.
Parameters
`namespaceURI` of type `DOMString` [p.21]
The *namespace URI* [p.450] of the attribute to retrieve.

`localName` of type `DOMString`
The *local name* [p.450] of the attribute to retrieve.

Return Value

`Attr` [p.56]	The `Attr` node with the specified attribute local name and namespace URI or `null` if there is no such attribute.

No Exceptions

`getElementsByTagName`
Returns a `NodeList` [p.47] of all descendant `Elements` with a given tag name, in the order in which they are encountered in a preorder traversal of this `Element` tree.
Parameters
`name` of type `DOMString` [p.21]
The name of the tag to match on. The special value "*" matches all tags.

Return Value

`NodeList` [p.47]	A list of matching `Element` nodes.

No Exceptions

`getElementsByTagNameNS` introduced in **DOM Level 2**
Returns a `NodeList` [p.47] of all the descendant `Elements` with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this `Element` tree.
HTML-only DOM implementations do not need to implement this method.
Parameters

`namespaceURI` of type `DOMString` [p.21]
: The *namespace URI* [p.450] of the elements to match on. The special value "*" matches all namespaces.

`localName` of type `DOMString`
: The *local name* [p.450] of the elements to match on. The special value "*" matches all local names.

Return Value

`NodeList` [p.47]	A new `NodeList` object containing all the matched `Elements`.

No Exceptions

`hasAttribute` introduced in **DOM Level 2**

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

Parameters

`name` of type `DOMString` [p.21]
: The name of the attribute to look for.

Return Value

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

No Exceptions

`hasAttributeNS` introduced in **DOM Level 2**

Returns `true` when an attribute with a given local name and namespace URI is specified on this element or has a default value, `false` otherwise. HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]
: The *namespace URI* [p.450] of the attribute to look for.

`localName` of type `DOMString`
: The *local name* [p.450] of the attribute to look for.

Return Value

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

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 as well as the corresponding namespace URI, local name, and prefix when applicable.
To remove an attribute by local name and namespace URI, use the `removeAttributeNS` method.

Parameters

`name` of type `DOMString` [p.21]
The name of the attribute to remove.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`removeAttributeNS` introduced in **DOM Level 2**

Removes an attribute by local name and namespace URI. If the removed attribute has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix.
HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]
The *namespace URI* [p.450] of the attribute to remove.

`localName` of type `DOMString`
The *local name* [p.450] of the attribute to remove.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

No Return Value

`removeAttributeNode`

Removes the specified attribute node. If the removed `Attr` [p.56] has a default value it is immediately replaced. The replacing attribute has the same namespace URI and local name, as well as the original prefix, when applicable.

Parameters

`oldAttr` of type `Attr` [p.56]
The `Attr` node to remove from the attribute list.

Return Value

`Attr` [p.56]	The `Attr` node that was removed.

Exceptions

`DOMException` [p.24]	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.56] node plus any `Text` [p.66] and `EntityReference` [p.70] nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.
To set an attribute with a qualified name and namespace URI, use the `setAttributeNS` method.

Parameters

`name` of type `DOMString` [p.21]
The name of the attribute to create or alter.

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

Exceptions

`DOMException` [p.24]	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

`setAttributeNS` introduced in **DOM Level 2**

Adds a new attribute. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the `qualifiedName`, and its value is changed to be the `value` parameter. This value is a simple string; it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately

escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` [p.56] node plus any `Text` [p.66] and `EntityReference` [p.70] nodes, build the appropriate subtree, and use `setAttributeNodeNS` or `setAttributeNode` to assign it as the value of an attribute.
HTML-only DOM implementations do not need to implement this method.

Parameters

`namespaceURI` of type `DOMString` [p.21]
The *namespace URI* [p.450] of the attribute to create or alter.

`qualifiedName` of type `DOMString`
The *qualified name* [p.450] of the attribute to create or alter.

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

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: Raised if the specified qualified name contains an illegal character.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.
	NAMESPACE_ERR: Raised if the `qualifiedName` is malformed, if the `qualifiedName` has a prefix and the `namespaceURI` is `null` or an empty string, if the `qualifiedName` has a prefix that is "xml" and the `namespaceURI` is different from "http://www.w3.org/XML/1998/namespace", or if the `qualifiedName` is "xmlns" and the `namespaceURI` is different from "http://www.w3.org/2000/xmlns/".

No Return Value

`setAttributeNode`
Adds a new attribute node. If an attribute with that name (`nodeName`) is already present in the element, it is replaced by the new one.
To add a new attribute node with a qualified name and namespace URI, use the `setAttributeNodeNS` method.

Parameters

`newAttr` of type `Attr` [p.56]
The `Attr` node to add to the attribute list.

Return Value

`Attr` [p.56]	If the `newAttr` attribute replaces an existing attribute, the replaced `Attr` node is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.24]	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.56] nodes to re-use them in other elements.

`setAttributeNodeNS` introduced in **DOM Level 2**

Adds a new attribute. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one.
HTML-only DOM implementations do not need to implement this method.

Parameters

`newAttr` of type `Attr` [p.56]
The `Attr` node to add to the attribute list.

Return Value

`Attr` [p.56]	If the `newAttr` attribute replaces an existing attribute with the same *local name* [p.450] and *namespace URI* [p.450] , the replaced `Attr` node is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.24]	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.56] nodes to re-use them in other elements.

Interface *Text*

The `Text` interface inherits from `CharacterData` [p.52] and represents the textual content (termed *character data* in XML) of an `Element` [p.57] or `Attr` [p.56] . 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.449] (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 `Node` [p.38] 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.447] offset at which to split, starting from `0`.

Return Value

`Text` [p.66]	The new node, of the same type as this node.

Exceptions

`DOMException` [p.24]	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.52] 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 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.26] interface to determine whether they are supported or not. The feature string for all the interfaces listed in this section is "XML".

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.21] attribute of the `Text` [p.66] 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.52] interface through the `Text` [p.66] interface. Adjacent `CDATASections` nodes are not merged by use of the `normalize` method of the `Node` [p.38] 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.29] has a `doctype` attribute whose value is either `null` or a `DocumentType` object. The `DocumentType` interface in the DOM 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 schema efforts on DTD representation are not clearly understood as of this writing.

The DOM Level 2 doesn't support editing `DocumentType` nodes.

IDL Definition

```
interface DocumentType : Node {
  readonly attribute DOMString        name;
  readonly attribute NamedNodeMap     entities;
  readonly attribute NamedNodeMap     notations;
  // Introduced in DOM Level 2:
  readonly attribute DOMString        publicId;
  // Introduced in DOM Level 2:
  readonly attribute DOMString        systemId;
  // Introduced in DOM Level 2:
  readonly attribute DOMString        internalSubset;
};
```

Attributes

`entities` of type `NamedNodeMap` [p.48] , readonly

A `NamedNodeMap` [p.48] 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.69] interface.
The DOM Level 2 does not support editing entities, therefore `entities` cannot be altered in any way.

`internalSubset` of type `DOMString` [p.21] , readonly, introduced in **DOM Level 2**

The internal subset as a string.

Note: The actual content returned depends on how much information is available to the implementation. This may vary depending on various parameters, including the XML processor used to build the document.

`name` of type `DOMString` [p.21] , readonly
: The name of DTD; i.e., the name immediately following the `DOCTYPE` keyword.

`notations` of type `NamedNodeMap` [p.48] , readonly
: A `NamedNodeMap` [p.48] containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the `Notation` [p.69] interface.
The DOM Level 2 does not support editing notations, therefore `notations` cannot be altered in any way.

`publicId` of type `DOMString` [p.21] , readonly, introduced in **DOM Level 2**
: The public identifier of the external subset.

`systemId` of type `DOMString` [p.21] , readonly, introduced in **DOM Level 2**
: The system identifier of the external subset.

Interface *Notation*

This interface represents a notation declared in the DTD. A notation either declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification), or is used for formal declaration of processing instruction targets (see section 2.6 of the XML 1.0 specification). The `nodeName` attribute inherited from `Node` [p.38] is set to the declared name of the notation.

The DOM Level 1 does not support editing `Notation` nodes; they are therefore *readonly* [p.450] .

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.21] , readonly
: The public identifier of this notation. If the public identifier was not specified, this is `null`.

`systemId` of type `DOMString` [p.21] , 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.38] 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.70] 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 DOM Level 2 does not support editing `Entity` nodes; if a user wants to make changes to the contents of an `Entity`, every related `EntityReference` [p.70] 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.450] .

An `Entity` node does not have any parent.

Note: If the entity contains an unbound *namespace prefix* [p.450] , the `namespaceURI` of the corresponding node in the `Entity` node subtree is `null`. The same is true for `EntityReference` [p.70] nodes that refer to this entity, when they are created using the `createEntityReference` method of the `Document` [p.29] interface. The DOM Level 2 does not support any mechanism to resolve namespace prefixes.

IDL Definition

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

Attributes

`notationName` of type `DOMString` [p.21] , readonly
: For unparsed entities, the name of the notation for the entity. For parsed entities, this is `null`.

`publicId` of type `DOMString` [p.21] , 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.21] , 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.69] node representing the referenced entity. If such an `Entity` exists, then the subtree of the `EntityReference` node is in general a copy of the `Entity` node subtree. However, this may not be true when an entity contains an unbound *namespace prefix* [p.450] . In such a case, because the namespace prefix resolution depends on where the entity reference is, the descendants of the `EntityReference` node may be bound to different *namespace URIs* [p.450] .

As for `Entity` [p.69] nodes, `EntityReference` nodes and all their descendants are *readonly* [p.450] .

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.21]

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.24]	NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

`target` of type `DOMString` [p.21] , 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
Arnaud Le Hors, W3C
Mike Champion, ArborText (for DOM Level 1)
Vidur Apparao, Netscape (for DOM Level 1)
Scott Isaacs, Microsoft (for DOM Level 1 until January 1998)
Chris Wilson, Microsoft (for DOM Level 1 after January 1998)
Ian Jacobs, W3C (for DOM Level 1)

2.1. Introduction

This section extends the 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.448] .
- 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` [p.76] interface, derived from the core `Document` [p.29] interface. `HTMLDocument` specifies the operations and queries that can be made on a HTML document.
- An `HTMLElement` [p.80] interface, derived from the core `Element` [p.57] 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.80] interface. For all such attributes, the derived interface for the element contains explicit methods for setting and getting the values.

The DOM Level 2 includes mechanisms to access and modify style specified through CSS and defines an event model that can be used with HTML documents.

The interfaces found within this section are not mandatory. A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether they are supported or not. The feature string for all the interfaces listed in this section is "HTML".

The interfaces in this specification are designed for HTML 4.0 documents, and not for XHTML 1.0 documents. Use of the HTML DOM with XHTML 1.0 documents may result in incorrect processing; see Appendix C11 in [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.448] implementations.

2.3. Miscellaneous Object Definitions

Interface *HTMLDOMImplementation* (introduced in **DOM Level 2**)

The `HTMLDOMImplementation` interface extends the `DOMImplementation` [p.26] interface with a method for creating an HTML document instance.

IDL Definition

```
// Introduced in DOM Level 2:
interface HTMLDOMImplementation : DOMImplementation {
  HTMLDocument       createHTMLDocument(in DOMString title);
};
```

Methods

`createHTMLDocument`

Creates an `HTMLDocument` [p.76] object with the minimal tree made of the following elements: `HTML`, `HEAD`, `TITLE`, and `BODY`.

Parameters

`title` of type `DOMString` [p.21]

The title of the document to be set as the content of the `TITLE` element, through a child `Text` [p.66] node.

Return Value

`HTMLDocument` [p.76]	A new `HTMLDocument` object.

No Exceptions

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.38]	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.38] 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.21]

The name of the `Node` [p.38] to be fetched.

Return Value

`Node` [p.38]	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

Note: In DOM Level 2, the method `getElementById` is inherited from the `Document` [p.29] interface where it was moved.

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);
  NodeList           getElementsByName(in DOMString elementName);
};
```

Attributes

`URL` of type `DOMString` [p.21] , readonly
The complete URI [RFC2396] of the document.

`anchors` of type `HTMLCollection` [p.75] , 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.75] , readonly
A collection of all the `OBJECT` elements that include applets and `APPLET` (*deprecated*) elements in a document.

`body` of type `HTMLElement` [p.80]
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.21]
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.21] , 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.75] , readonly
A collection of all the forms of a document.

`images` of type `HTMLCollection` [p.75] , 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.75] , 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.21] , readonly
: Returns the URI [RFC2396] 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.21]
: 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

`getElementsByName`
: Returns the (possibly empty) collection of elements whose `name` value is given by `elementName`.
Parameters
`elementName` of type `DOMString` [p.21]
The `name` attribute value for an element.

Return Value

`NodeList` [p.47]	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.21]
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.21]

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.448] . 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` [p.57] 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.

The return value of an attribute that is unspecified and does not have a default value is the empty string if the return type is a DOMString, false if the return type is a boolean and 0 if the return type is a number.

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 of an HTML element is accessible through the `ElementCSSInlineStyle` [p.158] interface which is defined in the Document Object Model CSS [p.133] .

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.21]
: 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.21]
: 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.21]
: The element’s identifier. See the id attribute definition in HTML 4.0.

`lang` of type `DOMString` [p.21]
: Language code defined in RFC 1766. See the lang attribute definition in HTML 4.0.

`title` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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 (see also the `LinkStyle` [p.131] interface in the Document Object Model StyleSheets [p.127] module).

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.21]
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.21]
The URI of the linked resource. See the href attribute definition in HTML 4.0.

`hreflang` of type `DOMString` [p.21]
Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

`media` of type `DOMString` [p.21]
Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

`rel` of type `DOMString` [p.21]
Forward link type. See the rel attribute definition in HTML 4.0.

`rev` of type `DOMString` [p.21]
Reverse link type. See the rev attribute definition in HTML 4.0.

`target` of type `DOMString` [p.21]
Frame to render the resource in. See the target attribute definition in HTML 4.0.

`type` of type `DOMString` [p.21]
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.21]
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.21]
Associated information. See the content attribute definition in HTML 4.0.

`httpEquiv` of type `DOMString` [p.21]
HTTP response header name. See the http-equiv attribute definition in HTML 4.0.

`name` of type `DOMString` [p.21]
Meta information name. See the name attribute definition in HTML 4.0.

`scheme` of type `DOMString` [p.21]
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.21]
The base URI. See the href attribute definition in HTML 4.0.

`target` of type `DOMString` [p.21]
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.85] , 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.21]
: The prompt message. See the prompt attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLStyleElement*

Style information. See the STYLE element definition in HTML 4.0, the Document Object Model CSS [p.133] module and the `LinkStyle` [p.131] interface in the Document Object Model StyleSheets [p.127] module.

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.21]
: Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

`type` of type `DOMString` [p.21]
: The content type pf 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: List of character sets supported by the server. See the accept-charset attribute definition in HTML 4.0.

`action` of type `DOMString` [p.21]
: Server-side form handler. See the action attribute definition in HTML 4.0.

`elements` of type `HTMLCollection` [p.75] , readonly
: Returns a collection of all control elements in the form.

`enctype` of type `DOMString` [p.21]
: 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.21]
: HTTP method used to submit form. See the method attribute definition in HTML 4.0.

`name` of type `DOMString` [p.21]
: Names the form.

`target` of type `DOMString` [p.21]
: 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.85] , 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.21]
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`options` of type `HTMLCollection` [p.75] , 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.21] , 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.21]
: The current form control value.

Methods

`add`
: Add a new element to the collection of `OPTION` elements for this `SELECT`. This method is the equivalent of the `appendChild` method of the `Node` [p.38] interface if the `before` parameter is `null`. It is equivalent to the `insertBefore` method on the parent of `before` in all other cases.

Parameters

`element` of type `HTMLElement` [p.80]
: The element to add.

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

Exceptions

`DOMException` [p.24]	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.21]

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.85] , 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.21]
: 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.21] , readonly
: The text contained within the option element.

`value` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.85] , 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.21]
: 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.21]
: 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.21]
: 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.21] , readonly
: The type of control created. See the type attribute definition in HTML 4.0.

`useMap` of type `DOMString` [p.21]
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`value` of type `DOMString` [p.21]
: 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.21]
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.21]
: 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.85] , 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.21]
: 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.21] , readonly
: The type of this form control. This the string "textarea".

`value` of type `DOMString` [p.21]
: 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.21] , 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.21]
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.85] , 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.21]
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.21] , readonly
: The type of button. See the type attribute definition in HTML 4.0.

`value` of type `DOMString` [p.21]
: 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.21]
: 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.85] , 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.21]
: 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.85] , 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.21]
: 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.21]
: 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.85] , 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.21]
: 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.21]
: 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.21]

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.21]

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.21]

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.21]
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.21]
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.21]
: 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.21]
: Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`face` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`face` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

`coords` of type `DOMString` [p.21]
: 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.21]
: The URI of the linked resource. See the href attribute definition in HTML 4.0.

`hreflang` of type `DOMString` [p.21]
: Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

`name` of type `DOMString` [p.21]
: Anchor name. See the name attribute definition in HTML 4.0.

`rel` of type `DOMString` [p.21]
: Forward link type. See the rel attribute definition in HTML 4.0.

`rev` of type `DOMString` [p.21]
: Reverse link type. See the rev attribute definition in HTML 4.0.

`shape` of type `DOMString` [p.21]
: 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.21]
: Frame to render the resource in. See the target attribute definition in HTML 4.0.

`type` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: Override height. See the height attribute definition in HTML 4.0.

`hspace` of type `DOMString` [p.21]
: 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.21]
: URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`lowSrc` of type `DOMString` [p.21]
: URI designating the source of this image, for low-resolution output.

`name` of type `DOMString` [p.21]
: The name of the element (for backwards compatibility).

`src` of type `DOMString` [p.21]
: URI designating the source of this image. See the src attribute definition in HTML 4.0.

`useMap` of type `DOMString` [p.21]
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`vspace` of type `DOMString` [p.21]
: 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.21]
: 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;
  // Introduced in DOM Level 2:
  readonly attribute Document         contentDocument;
};
```

Attributes

`align` of type `DOMString` [p.21]
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.21]
Space-separated list of archives. See the archive attribute definition in HTML 4.0.

`border` of type `DOMString` [p.21]
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.21]
Applet class file. See the `code` attribute for HTMLAppletElement.

`codeBase` of type `DOMString` [p.21]
Base URI for `classid`, `data`, and `archive` attributes. See the codebase attribute definition in HTML 4.0.

`codeType` of type `DOMString` [p.21]
Content type for data downloaded via `classid` attribute. See the codetype attribute definition in HTML 4.0.

`contentDocument` of type `Document` [p.29] , readonly, introduced in **DOM Level 2**
: The document this object contains, if there is any and it is available, or `null` otherwise.

`data` of type `DOMString` [p.21]
: 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.85] , 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.21]
: Override height. See the height attribute definition in HTML 4.0.

`hspace` of type `DOMString` [p.21]
: 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.21]
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`standby` of type `DOMString` [p.21]
: 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.21]
: Content type for data downloaded via `data` attribute. See the type attribute definition in HTML 4.0.

`useMap` of type `DOMString` [p.21]
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`vspace` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: The name of a run-time parameter. See the name attribute definition in HTML 4.0.

`type` of type `DOMString` [p.21]
: 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.21]
: The value of a run-time parameter. See the value attribute definition in HTML 4.0.

`valueType` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: Override height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`hspace` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.75] , readonly
: The list of areas defined for the image map.

`name` of type `DOMString` [p.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
: 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.21]
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.21]
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.21]
Reserved for future use.

`htmlFor` of type `DOMString` [p.21]
Reserved for future use.

`src` of type `DOMString` [p.21]
URI designating an external script. See the src attribute definition in HTML 4.0.

`text` of type `DOMString` [p.21]
The script content of the element.

`type` of type `DOMString` [p.21]
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.21]
: 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.21]
: 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.21]
: The width of the border around the table. See the border attribute definition in HTML 4.0.

`caption` of type `HTMLTableCaptionElement` [p.115]
: Returns the table's `CAPTION`, or void if none exists.

`cellPadding` of type `DOMString` [p.21]
: 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.21]
: Specifies the horizontal and vertical separation between cells. See the cellspacing attribute definition in HTML 4.0.

`frame` of type `DOMString` [p.21]
: Specifies which external table borders to render. See the frame attribute definition in HTML 4.0.

`rows` of type `HTMLCollection` [p.75] , readonly
: Returns a collection of all the rows in the table, including all in `THEAD`, `TFOOT`, all `TBODY` elements.

`rules` of type `DOMString` [p.21]
: Specifies which internal table borders to render. See the rules attribute definition in HTML 4.0.

`summary` of type `DOMString` [p.21]
: Description about the purpose or structure of a table. See the summary attribute definition in HTML 4.0.

`tBodies` of type `HTMLCollection` [p.75] , readonly
: Returns a collection of the defined table bodies.

`tFoot` of type `HTMLTableSectionElement` [p.116]
: Returns the table's `TFOOT`, or `null` if none exists.

`tHead` of type `HTMLTableSectionElement` [p.116]
: Returns the table's `THEAD`, or `null` if none exists.

`width` of type `DOMString` [p.21]
: 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.80]	A `CAPTION` element.

No Parameters
No Exceptions

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

Return Value

`HTMLElement` [p.80]	A footer element (`TFOOT`).

No Parameters
No Exceptions

`createTHead`

Create a table header row or return an existing one.

Return Value

`HTMLElement` [p.80]	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.24]	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.80]	The newly created row.

Exceptions

`DOMException` [p.24]	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.21]

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.21]

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

`ch` of type `DOMString` [p.21]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.21]
: 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.21]
: Vertical alignment of cell data in column. See the valign attribute definition in HTML 4.0.

`width` of type `DOMString` [p.21]
: 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.21]
: Horizontal alignment of data in cells. See the `align` attribute for HTMLTheadElement for details.

`ch` of type `DOMString` [p.21]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.21]
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`rows` of type `HTMLCollection` [p.75] , readonly
: The collection of rows in this table section.

`vAlign` of type `DOMString` [p.21]
: 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.24]	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.80]	The newly created row.

Exceptions

`DOMException` [p.24]	INDEX_SIZE_ERR: Raised if the specified index is greater than the number of rows of if the index is neagative.

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.21]
: Horizontal alignment of data within cells of this row. See the align attribute definition in HTML 4.0.

`bgColor` of type `DOMString` [p.21]
: 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.75] , readonly
: The collection of cells in this row.

`ch` of type `DOMString` [p.21]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.21]
: 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.21]
: 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.24]	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.80]	The newly created cell.

Exceptions

`DOMException` [p.24]	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.21]

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

`align` of type `DOMString` [p.21]
: Horizontal alignment of data in cell. See the align attribute definition in HTML 4.0.

`axis` of type `DOMString` [p.21]
: Names group of related headers. See the axis attribute definition in HTML 4.0.

`bgColor` of type `DOMString` [p.21]
: 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.21]
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff` of type `DOMString` [p.21]
: 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.21]
: List of `id` attribute values for header cells. See the headers attribute definition in HTML 4.0.

`height` of type `DOMString` [p.21]
: 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.21]
: Scope covered by header cells. See the scope attribute definition in HTML 4.0.

`vAlign` of type `DOMString` [p.21]
: Vertical alignment of data in cell. See the valign attribute definition in HTML 4.0.

`width` of type `DOMString` [p.21]
: 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.21]
The number of columns of frames in the frameset. See the cols attribute definition in HTML 4.0.

`rows` of type `DOMString` [p.21]
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;
  // Introduced in DOM Level 2:
  readonly attribute Document         contentDocument;
};
```

Attributes

`contentDocument` of type `Document` [p.29] , readonly, introduced in **DOM Level 2**
The document this frame contains, if there is any and it is available, or `null` otherwise.

`frameBorder` of type `DOMString` [p.21]
Request frame borders. See the frameborder attribute definition in HTML 4.0.

`longDesc` of type `DOMString` [p.21]
URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`marginHeight` of type `DOMString` [p.21]
Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

`marginWidth` of type `DOMString` [p.21]
: Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

`name` of type `DOMString` [p.21]
: 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.21]
: Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

`src` of type `DOMString` [p.21]
: 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;
  // Introduced in DOM Level 2:
  readonly attribute Document         contentDocument;
};
```

Attributes

`align` of type `DOMString` [p.21]
: 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.

`contentDocument` of type `Document` [p.29] , readonly, introduced in **DOM Level 2**
: The document this frame contains, if there is any and it is available, or `null` otherwise.

`frameBorder` of type `DOMString` [p.21]
: Request frame borders. See the frameborder attribute definition in HTML 4.0.

`height` of type `DOMString` [p.21]
: Frame height. See the height attribute definition in HTML 4.0.

`longDesc` of type `DOMString` [p.21]
: URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`marginHeight` of type `DOMString` [p.21]
: Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

`marginWidth` of type `DOMString` [p.21]
: Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

`name` of type `DOMString` [p.21]
: The frame name (object of the `target` attribute). See the name attribute definition in HTML 4.0.

`scrolling` of type `DOMString` [p.21]
: Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

`src` of type `DOMString` [p.21]
: A URI designating the initial frame contents. See the src attribute definition in HTML 4.0.

`width` of type `DOMString` [p.21]
: Frame width. See the width attribute definition in HTML 4.0.

2.5.5. Object definitions

3. Document Object Model Views

Editors
Laurence Cable, Sun
Arnaud Le Hors, W3C

3.1. Introduction

A document may have one or more "views" associated with it, e.g., a computed view on a document after applying a CSS stylesheet, or multiple presentations (e.g., HTML Frame) of the same document in a client. That is, a view is some alternate representation of, or a presentation of, and associated with, a source document.

A view may be static, reflecting the state of the document when the view was created, or dynamic, reflecting changes in the target document as they occur, subsequent to the view being created. This Level of the DOM specification makes no statement about these behaviors.

This section defines an `AbstractView` [p.125] interface which provides a base interface from which all such views shall derive. It defines an attribute which references the target document of the `AbstractView`. The only semantics of the `AbstractView` defined here create an association between a view and its target document.

There are no subinterfaces of `AbstractView` [p.125] defined in the DOM Level 2.

However, `AbstractView` [p.125] is defined in and used in this Level in two places:

- A Document may implement a `DocumentView` [p.126] that has a default view attribute associated with it. This default view is typically dependent on the implementation (e.g., the browser frame rendering the document). The default view can be used in order to identify and/or associate a view with its target document (by testing object equality on the `AbstractView` [p.125] or obtaining the `DocumentView` attribute).
- A `UIEvent` [p.231] typically occurs upon a view of a Document (e.g., a mouse click on a browser frame rendering a particular Document instance). A `UIEvent` has an `AbstractView` [p.125] associated with it which identifies both the particular (implementation-dependent) view in which the event occurs, and the target document the `UIEvent` is related to.

The interfaces found within this section are not mandatory. A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether they are supported or not. The feature string for all the interfaces listed in this section is "Views".

3.2. Interfaces

Interface *AbstractView* (introduced in **DOM Level 2**)

A base interface that all views shall derive from.

IDL Definition

```
// Introduced in DOM Level 2:
interface AbstractView {
  readonly attribute DocumentView     document;
};
```

Attributes

`document` of type `DocumentView` [p.126] , readonly

The source `DocumentView` [p.126] of which this is an `AbstractView`.

Interface *DocumentView* (introduced in **DOM Level 2**)

The `DocumentView` interface is implemented by `Document` [p.29] objects in DOM implementations supporting DOM Views. It provides an attribute to retrieve the default view of a document.

IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentView {
  readonly attribute AbstractView     defaultView;
};
```

Attributes

`defaultView` of type `AbstractView` [p.125] , readonly

The default `AbstractView` [p.125] for this `Document` [p.29] , or `null` if none available.

4. Document Object Model StyleSheets

Editors

Vidur Apparao, Netscape Communications Corp.
Philippe Le Hégaret, W3C
Chris Wilson, Microsoft

4.1. Introduction

The DOM Level 2 Style Sheet interfaces are base interfaces used to represent any type of style sheet. The expectation is that DOM modules that represent a specific style sheet language may contain interfaces that derive from these interfaces.

A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether the StyleSheets interfaces are supported or not. The feature string for the fundamental interfaces listed in this section is "StyleSheets".

4.2. Style Sheet Interfaces

This set of interfaces represents the generic notion of style sheets.

Interface *StyleSheet* (introduced in **DOM Level 2**)

The `StyleSheet` interface is the abstract base interface for any type of style sheet. It represents a single style sheet associated with a structured document. In HTML, the StyleSheet interface represents either an external style sheet, included via the HTML *LINK* element, or an inline *STYLE* element. In XML, this interface represents an external style sheet, included via a *style sheet processing instruction*.

IDL Definition

```
// Introduced in DOM Level 2:
interface StyleSheet {
  readonly attribute DOMString        type;
           attribute boolean          disabled;
  readonly attribute Node             ownerNode;
  readonly attribute StyleSheet       parentStyleSheet;
  readonly attribute DOMString        href;
  readonly attribute DOMString        title;
  readonly attribute MediaList        media;
};
```

Attributes

`disabled` of type `boolean`

`false` if the style sheet is applied to the document. `true` if it is not. Modifying this attribute may cause a new resolution of style for the document. A stylesheet only applies if both an appropriate medium definition is present and the disabled attribute is false. So, if the media doesn't apply to the current user agent, the `disabled` attribute is ignored.

`href` of type `DOMString` [p.21] , readonly
: If the style sheet is a linked style sheet, the value of its attribute is its location. For inline style sheets, the value of this attribute is `null`. See the *href attribute definition* for the `LINK` element in HTML 4.0, and the href pseudo-attribute for the XML *style sheet processing instruction*.

`media` of type `MediaList` [p.129] , readonly
: The intended destination media for style information. The media is often specified in the `ownerNode`. If no media has been specified, the `MediaList` [p.129] will be empty. See the *media attribute definition* for the `LINK` element in HTML 4.0, and the media pseudo-attribute for the XML *style sheet processing instruction* . Modifying the media list may cause a change to the attribute `disabled`.

`ownerNode` of type `Node` [p.38] , readonly
: The node that associates this style sheet with the document. For HTML, this may be the corresponding `LINK` or `STYLE` element. For XML, it may be the linking processing instruction. For style sheets that are included by other style sheets, the value of this attribute is `null`.

`parentStyleSheet` of type `StyleSheet` [p.127] , readonly
: For style sheet languages that support the concept of style sheet inclusion, this attribute represents the including style sheet, if one exists. If the style sheet is a top-level style sheet, or the style sheet language does not support inclusion, the value of this attribute is `null`.

`title` of type `DOMString` [p.21] , readonly
: The advisory title. The title is often specified in the `ownerNode`. See the *title attribute definition* for the `LINK` element in HTML 4.0, and the title pseudo-attribute for the XML *style sheet processing instruction*.

`type` of type `DOMString` [p.21] , readonly
: This specifies the style sheet language for this style sheet. The style sheet language is specified as a content type (e.g. "text/css"). The *content type* is often specified in the `ownerNode`. Also see the *type attribute definition* for the `LINK` element in HTML 4.0, and the type pseudo-attribute for the XML *style sheet processing instruction*.

Interface *StyleSheetList* (introduced in **DOM Level 2**)

The `StyleSheetList` interface provides the abstraction of an ordered collection of style sheets.
IDL Definition

```
// Introduced in DOM Level 2:
interface StyleSheetList {
  readonly attribute unsigned long    length;
  StyleSheet         item(in unsigned long index);
};
```

Attributes

`length` of type `unsigned long`, readonly
The number of `StyleSheets` [p.127] in the list. The range of valid child stylesheet indices is `0` to `length-1` inclusive.

Methods

`item`
Used to retrieve a style sheet by ordinal index.
Parameters
`index` of type `unsigned long`
Index into the collection

Return Value

`StyleSheet` [p.127]	The style sheet at the `index` position in the `StyleSheetList`, or `null` if that is not a valid index.

No Exceptions

Interface *MediaList* (introduced in **DOM Level 2**)

The `MediaList` interface provides the abstraction of an ordered collection of *media*, without defining or constraining how this collection is implemented. An empty list is the same as a list that contains the medium `"all"`.
IDL Definition

```
// Introduced in DOM Level 2:
interface MediaList {
           attribute DOMString        mediaText;
                                        // raises(DOMException) on setting

  readonly attribute unsigned long    length;
  DOMString          item(in unsigned long index);
  void               deleteMedium(in DOMString oldMedium)
                                        raises(DOMException);
  void               appendMedium(in DOMString newMedium)
                                        raises(DOMException);
};
```

Attributes

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

`mediaText` of type `DOMString` [p.21]
The parsable textual representation of the media list. This is a comma-separated list of media.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this media list is readonly.

Methods

`appendMedium`

Adds the medium `newMedium` to the end of the list. If the `newMedium` is already used, it is first removed.

Parameters

`newMedium` of type `DOMString` [p.21]

The new medium to add.

Exceptions

`DOMException` [p.24]	INVALID_CHARACTER_ERR: If the medium contains characters that are invalid in the underlying style language. NO_MODIFICATION_ALLOWED_ERR: Raised if this list is readonly.

No Return Value

`deleteMedium`

Deletes the medium indicated by `oldMedium` from the list.

Parameters

`oldMedium` of type `DOMString` [p.21]

The medium to delete in the media list.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this list is readonly. NOT_FOUND_ERR: Raised if `oldMedium` is not in the list.

No Return Value

`item`

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

Parameters

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

Return Value

`DOMString` [p.21]	The medium at the `index`th position in the `MediaList`, or `null` if that is not a valid index.

No Exceptions

4.3. Document Extensions

Interface *LinkStyle* (introduced in **DOM Level 2**)

The `LinkStyle` interface provides a mechanism by which a style sheet can be retrieved from the node responsible for linking it into a document. An instance of the `LinkStyle` interface can be obtained using binding-specific casting methods on an instance of a linking node (`HTMLLinkElement` [p.81] , `HTMLStyleElement` [p.84] or `ProcessingInstruction` [p.71] in DOM Level 2).

IDL Definition

```
// Introduced in DOM Level 2:
interface LinkStyle {
  readonly attribute StyleSheet       sheet;
};
```

Attributes

`sheet` of type `StyleSheet` [p.127] , readonly
The style sheet.

Interface *DocumentStyle* (introduced in **DOM Level 2**)

The `DocumentStyle` interface provides a mechanism by which the style sheets embedded in a document can be retrieved. The expectation is that an instance of the `DocumentStyle` interface can be obtained by using binding-specific casting methods on an instance of the `Document` [p.29] interface.

IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentStyle {
  readonly attribute StyleSheetList   styleSheets;
};
```

Attributes

`styleSheets` of type `StyleSheetList` [p.128] , readonly
A list containing all the style sheets explicitly linked into or embedded in a document. For HTML documents, this includes external style sheets, included via the HTML *LINK* element, and inline *STYLE* elements. In XML, this includes external style sheets, included via style sheet processing instructions (see [XML-StyleSheet]).

4.4. Association between a style sheet and a document.

HTML and Style Sheet Creation

A style sheet can be associated with an HTMLDocument in one of two ways:

- By creating a new LINK HTML element (see the `HTMLLinkElement` [p.81] interface). The underlying style sheet will be created after the element is inserted into the document and both the href and the type attribute have been set in a way indicating that the linked object is a style sheet.
- By creating a new STYLE HTML element (see the `HTMLStyleElement` [p.84] interface). The underlying style sheet will be created after the element is inserted into the document and the type attribute is set in a way indicating that the element corresponds to a style sheet language interpreted by the user agent.

HTML and Style Sheet Removal

Removing a LINK HTML element or a STYLE HTML element removes the underlying style sheet from the style sheet collection associated with a document. Specifically, the removed style sheet is no longer applied to the presentation of the document.

XML and Style Sheet Creation

A new style sheet can be created and associated with an XML document by creating a processing instruction with the target ’xml-stylesheet’ [XML-StyleSheet] and inserting it into the document.

XML and Style Sheet Removal

Removing a processing instruction with a target of ’xml-stylesheet’ [XML-StyleSheet]removes the underlying style sheet from the style sheet collection associated with a document. Specifically, the removed style sheet is no longer applied to the presentation of the document.

5. Document Object Model CSS

Editors
Vidur Apparao, Netscape Communications Corp.
Philippe Le Hégaret, W3C
Chris Wilson, Microsoft

5.1. Overview of the DOM Level 2 CSS Interfaces

The DOM Level 2 Cascading Style Sheets (CSS) interfaces are designed with the goal of exposing CSS constructs to object model consumers. Cascading Style Sheets is a declarative syntax for defining presentation rules, properties and ancillary constructs used to format and render Web documents. This document specifies a mechanism to programmatically access and modify the rich style and presentation control provided by CSS (specifically CSS level 2 [CSS2]). This augments CSS by providing a mechanism to dynamically control the inclusion and exclusion of individual style sheets, as well as manipulate CSS rules and properties.

The CSS interfaces are organized in a logical, rather than physical structure. A collection of all style sheets referenced by or embedded in the document is accessible on the document interface. Each item in this collection exposes the properties common to all style sheets referenced or embedded in HTML and XML documents; this interface is described in the Document Object Model StyleSheets [p.127] . User style sheets are not accessible through this collection, in part due to potential privacy concerns (and certainly read-write issues).

For each CSS style sheet, an additional interface is exposed - the `CSSStyleSheet` [p.134] interface. This interface allows access to the collection of rules within a CSS style sheet and methods to modify that collection. Interfaces are provided for each specific type of rule in CSS2 (e.g. style declarations, `@import` rules, or `@font-face` rules), as well as a shared generic `CSSRule` [p.136] interface.

The most common type of rule is a style declaration. The `CSSStyleRule` [p.138] interface that represents this type of rule provides string access to the CSS selector of the rule, and access to the property declarations through the `CSSStyleDeclaration` [p.142] interface.

Finally, an optional `CSS2Properties` [p.181] interface is described; this interface (if implemented) provides shortcuts to the string values of all the properties in CSS level 2.

All CSS objects in the DOM are "live", that is, a change in the style sheet is reflected in the computed and actual style.

A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether the CSS interfaces are supported or not. The feature string for the CSS Model is "CSS". The existence within an implementation of the extended interfaces can also be queried using the `hasFeature` method.

5.2. CSS Fundamental Interfaces

The interfaces within this section are considered fundamental CSS interfaces, and must be supported by all conforming implementations of the CSS DOM module. These interfaces represent CSS style sheets specifically.

Interface *CSSStyleSheet* (introduced in **DOM Level 2**)

The `CSSStyleSheet` interface is a concrete interface used to represent a CSS style sheet i.e., a style sheet whose content type is "text/css".

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSStyleSheet : stylesheets::StyleSheet {
  readonly attribute CSSRule          ownerRule;
  readonly attribute CSSRuleList      cssRules;
  unsigned long      insertRule(in DOMString rule,
                                in unsigned long index)
                                        raises(DOMException);
  void               deleteRule(in unsigned long index)
                                        raises(DOMException);
};
```

Attributes

`cssRules` of type `CSSRuleList` [p.135] , readonly

The list of all CSS rules contained within the style sheet. This includes both *rule sets* and *at-rules*.

`ownerRule` of type `CSSRule` [p.136] , readonly

If this style sheet comes from an `@import` rule, the `ownerRule` attribute will contain the `CSSImportRule` [p.141] . In that case, the `ownerNode` attribute in the `StyleSheet` [p.127] interface will be `null`. If the style sheet comes from an element or a processing instruction, the `ownerRule` attribute will be `null` and the `ownerNode` attribute will contain the `Node` [p.38] .

Methods

`deleteRule`

Used to delete a rule from the style sheet.

Parameters

`index` of type `unsigned long`

The index within the style sheet's rule list of the rule to remove.

Exceptions

`DOMException` [p.24]	INDEX_SIZE_ERR: Raised if the specified index does not correspond to a rule in the style sheet's rule list. NO_MODIFICATION_ALLOWED_ERR: Raised if this style sheet is readonly.

No Return Value

`insertRule`

Used to insert a new rule into the style sheet. The new rule now becomes part of the cascade.

Parameters

`rule` of type `DOMString` [p.21]

The parsable text representing the rule. For rule sets this contains both the selector and the style declaration. For at-rules, this specifies both the at-identifier and the rule content.

`index` of type `unsigned long`

The index within the style sheet's rule list of the rule before which to insert the specified rule. If the specified index is equal to the length of the style sheet's rule collection, the rule will be added to the end of the style sheet.

Return Value

`unsigned long`	The index within the style sheet's rule collection of the newly inserted rule.

Exceptions

`DOMException` [p.24]	HIERARCHY_REQUEST_ERR: Raised if the rule cannot be inserted at the specified index e.g. if an `@import` rule is inserted after a standard rule set or other at-rule. INDEX_SIZE_ERR: Raised if the specified index is not a valid insertion point. NO_MODIFICATION_ALLOWED_ERR: Raised if this style sheet is readonly. SYNTAX_ERR: Raised if the specified rule has a syntax error and is unparsable.

Interface *CSSRuleList* (introduced in **DOM Level 2**)

The `CSSRuleList` interface provides the abstraction of an ordered collection of CSS rules.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSRuleList {
  readonly attribute unsigned long    length;
  CSSRule            item(in unsigned long index);
};
```

Attributes

`length` of type `unsigned long`, readonly
: The number of `CSSRules` [p.136] in the list. The range of valid child rule indices is `0` to `length-1` inclusive.

Methods

`item`
: Used to retrieve a CSS rule by ordinal index. The order in this collection represents the order of the rules in the CSS style sheet.

Parameters

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

Return Value

`CSSRule` [p.136]	The style rule at the `index` position in the `CSSRuleList`, or `null` if that is not a valid index.

No Exceptions

Interface *CSSRule* (introduced in **DOM Level 2**)

The `CSSRule` interface is the abstract base interface for any type of CSS *statement*. This includes both *rule sets* and *at-rules*. An implementation is expected to preserve all rules specified in a CSS style sheet, even if the rule is not recognized by the parser. Unrecognized rules are represented using the `CSSUnknownRule` [p.142] interface.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSRule {
  // RuleType
  const unsigned short      UNKNOWN_RULE                   = 0;
  const unsigned short      STYLE_RULE                     = 1;
  const unsigned short      CHARSET_RULE                   = 2;
  const unsigned short      IMPORT_RULE                    = 3;
  const unsigned short      MEDIA_RULE                     = 4;
  const unsigned short      FONT_FACE_RULE                 = 5;
  const unsigned short      PAGE_RULE                      = 6;

  readonly attribute unsigned short   type;
           attribute DOMString        cssText;
                                        // raises(DOMException) on setting

  readonly attribute CSSStyleSheet    parentStyleSheet;
  readonly attribute CSSRule          parentRule;
};
```

Definition group *RuleType*

An integer indicating which type of rule this is.

Defined Constants

`CHARSET_RULE`
: The rule is a `CSSCharsetRule` [p.141] .

`FONT_FACE_RULE`
: The rule is a `CSSFontFaceRule` [p.140] .

`IMPORT_RULE`
: The rule is a `CSSImportRule` [p.141] .

`MEDIA_RULE`
: The rule is a `CSSMediaRule` [p.138] .

`PAGE_RULE`
: The rule is a `CSSPageRule` [p.140] .

`STYLE_RULE`
: The rule is a `CSSStyleRule` [p.138] .

`UNKNOWN_RULE`
: The rule is a `CSSUnknownRule` [p.142] .

Attributes

`cssText` of type `DOMString` [p.21]

The parsable textual representation of the rule. This reflects the current state of the rule and not its initial value.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable.
	INVALID_MODIFICATION_ERR: Raised if the specified CSS string value represents a different type of rule than the current one.
	HIERARCHY_REQUEST_ERR: Raised if the rule cannot be inserted at this point in the style sheet.
	NO_MODIFICATION_ALLOWED_ERR: Raised if the rule is readonly.

`parentRule` of type `CSSRule` [p.136] , readonly

If this rule is contained inside another rule (e.g. a style rule inside an @media block), this is the containing rule. If this rule is not nested inside any other rules, this returns `null`.

`parentStyleSheet` of type `CSSStyleSheet` [p.134] , readonly
The style sheet that contains this rule.

`type` of type `unsigned short`, readonly
The type of the rule, as defined above. The expectation is that binding-specific casting methods can be used to cast down from an instance of the `CSSRule` interface to the specific derived interface implied by the `type`.

Interface *CSSStyleRule* (introduced in **DOM Level 2**)

The `CSSStyleRule` interface represents a single *rule set* in a CSS style sheet.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSStyleRule : CSSRule {
           attribute DOMString        selectorText;
                                        // raises(DOMException) on setting

  readonly attribute CSSStyleDeclaration  style;
};
```

Attributes
`selectorText` of type `DOMString` [p.21]
The textual representation of the *selector* for the rule set. The implementation may have stripped out insignificant whitespace while parsing the selector.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this rule is readonly.

`style` of type `CSSStyleDeclaration` [p.142] , readonly
The *declaration-block* of this rule set.

Interface *CSSMediaRule* (introduced in **DOM Level 2**)

The `CSSMediaRule` interface represents a *@media rule* in a CSS style sheet. A `@media` rule can be used to delimit style rules for specific media types.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSMediaRule : CSSRule {
  readonly attribute stylesheets::MediaList  media;
  readonly attribute CSSRuleList      cssRules;
  unsigned long      insertRule(in DOMString rule,
                                in unsigned long index)
                                        raises(DOMException);
  void               deleteRule(in unsigned long index)
                                        raises(DOMException);
};
```

Attributes

`cssRules` of type `CSSRuleList` [p.135] , readonly
: A list of all CSS rules contained within the media block.

`media` of type `stylesheets::MediaList`, readonly
: A list of *media types* for this rule.

Methods

`deleteRule`

Used to delete a rule from the media block.

Parameters

`index` of type `unsigned long`
: The index within the media block's rule collection of the rule to remove.

Exceptions

`DOMException` [p.24]	INDEX_SIZE_ERR: Raised if the specified index does not correspond to a rule in the media rule list.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this media rule is readonly.

No Return Value

`insertRule`

Used to insert a new rule into the media block.

Parameters

`rule` of type `DOMString` [p.21]
: The parsable text representing the rule. For rule sets this contains both the selector and the style declaration. For at-rules, this specifies both the at-identifier and the rule content.

`index` of type `unsigned long`
: The index within the media block's rule collection of the rule before which to insert the specified rule. If the specified index is equal to the length of the media blocks's rule collection, the rule will be added to the end of the media block.

Return Value

`unsigned long`	The index within the media block's rule collection of the newly inserted rule.

Exceptions

`DOMException` [p.24]	HIERARCHY_REQUEST_ERR: Raised if the rule cannot be inserted at the specified index, e.g., if an `@import` rule is inserted after a standard rule set or other at-rule. INDEX_SIZE_ERR: Raised if the specified index is not a valid insertion point. NO_MODIFICATION_ALLOWED_ERR: Raised if this media rule is readonly. SYNTAX_ERR: Raised if the specified rule has a syntax error and is unparsable.

Interface *CSSFontFaceRule* (introduced in **DOM Level 2**)

The `CSSFontFaceRule` interface represents a *@font-face rule* in a CSS style sheet. The `@font-face` rule is used to hold a set of font descriptions.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSFontFaceRule : CSSRule {
  readonly attribute CSSStyleDeclaration  style;
};
```

Attributes
`style` of type `CSSStyleDeclaration` [p.142] , readonly
The *declaration-block* of this rule.

Interface *CSSPageRule* (introduced in **DOM Level 2**)

The `CSSPageRule` interface represents a *@page rule* within a CSS style sheet. The `@page` rule is used to specify the dimensions, orientation, margins, etc. of a page box for paged media.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSPageRule : CSSRule {
           attribute DOMString        selectorText;
                                        // raises(DOMException) on setting

  readonly attribute CSSStyleDeclaration  style;
};
```

Attributes

`selectorText` of type `DOMString` [p.21]

The parsable textual representation of the page selector for the rule.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this rule is readonly.

`style` of type `CSSStyleDeclaration` [p.142] , readonly

The *declaration-block* of this rule.

Interface *CSSImportRule* (introduced in **DOM Level 2**)

The `CSSImportRule` interface represents a *@import rule* within a CSS style sheet. The `@import` rule is used to import style rules from other style sheets.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSImportRule : CSSRule {
  readonly attribute DOMString        href;
  readonly attribute stylesheets::MediaList  media;
  readonly attribute CSSStyleSheet    styleSheet;
};
```

Attributes

`href` of type `DOMString` [p.21] , readonly

The location of the style sheet to be imported. The attribute will not contain the `"url(...)"` specifier around the URI.

`media` of type `stylesheets::MediaList`, readonly

A list of media types for which this style sheet may be used.

`styleSheet` of type `CSSStyleSheet` [p.134] , readonly

The style sheet referred to by this rule, if it has been loaded. The value of this attribute is `null` if the style sheet has not yet been loaded or if it will not be loaded (e.g. if the style sheet is for a media type not supported by the user agent).

Interface *CSSCharsetRule* (introduced in **DOM Level 2**)

The `CSSCharsetRule` interface represents a *@charset rule* in a CSS style sheet. The value of the `encoding` attribute does not affect the encoding of text data in the DOM objects; this encoding is always UTF-16. After a stylesheet is loaded, the value of the `encoding` attribute is the value found in the `@charset` rule. If there was no `@charset` in the original document, then no `CSSCharsetRule` is created. The value of the `encoding` attribute may also be used as a hint for the encoding used on serialization of the style sheet.

The value of the *@charset rule* (and therefore of the `CSSCharsetRule`) may not correspond to the encoding the document actually came in; character encoding information e.g. in an HTTP header, has priority (see *CSS document representation*) but this is not reflected in the `CSSCharsetRule`.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSCharsetRule : CSSRule {
           attribute DOMString        encoding;
                                        // raises(DOMException) on setting

};
```

Attributes
`encoding` of type `DOMString` [p.21]
The encoding information used in this `@charset` rule.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified encoding value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this encoding rule is readonly.

Interface *CSSUnknownRule* (introduced in **DOM Level 2**)

The `CSSUnknownRule` interface represents an at-rule not supported by this user agent.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSUnknownRule : CSSRule {
};
```

Interface *CSSStyleDeclaration* (introduced in **DOM Level 2**)

The `CSSStyleDeclaration` interface represents a single *CSS declaration block*. This interface may be used to determine the style properties currently set in a block or to set style properties explicitly within the block.

While an implementation may not recognize all CSS properties within a CSS declaration block, it is expected to provide access to all specified properties in the style sheet through the `CSSStyleDeclaration` interface. Furthermore, implementations that support a specific level of CSS should correctly handle *CSS shorthand* properties for that level. For a further discussion of shorthand properties, see the `CSS2Properties` [p.181] interface.

This interface is also used to provide a **read-only** access to the *computed values* of an element. See also the `ViewCSS` [p.156] interface.

Note: The CSS Object Model doesn't provide an access to the *specified* or *actual* values of the CSS cascade.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSStyleDeclaration {
           attribute DOMString        cssText;
                                        // raises(DOMException) on setting

  DOMString          getPropertyValue(in DOMString propertyName);
  CSSValue           getPropertyCSSValue(in DOMString propertyName);
  DOMString          removeProperty(in DOMString propertyName)
                                        raises(DOMException);
  DOMString          getPropertyPriority(in DOMString propertyName);
  void               setProperty(in DOMString propertyName,
                                 in DOMString value,
                                 in DOMString priority)
                                        raises(DOMException);
  readonly attribute unsigned long    length;
  DOMString          item(in unsigned long index);
  readonly attribute CSSRule          parentRule;
};
```

Attributes

`cssText` of type `DOMString` [p.21]

The parsable textual representation of the declaration block (excluding the surrounding curly braces). Setting this attribute will result in the parsing of the new value and resetting of the properties in the declaration block. It also allows the insertion of additional properties and their values into the block.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly or a property is readonly.

`length` of type `unsigned long`, readonly

The number of properties that have been explicitly set in this declaration block. The range of valid indices is 0 to length-1 inclusive.

`parentRule` of type `CSSRule` [p.136] , readonly

The CSS rule that contains this declaration block or `null` if this `CSSStyleDeclaration` is not attached to a `CSSRule` [p.136] .

Methods

`getPropertyCSSValue`

Used to retrieve the object representation of the value of a CSS property if it has been explicitly set within this declaration block. This method returns `null` if the property is a *shorthand* property. Shorthand property values can only be accessed and modified as strings, using the `getPropertyValue` and `setProperty` methods.

Parameters

`propertyName` of type `DOMString` [p.21]

The name of the CSS property. See the *CSS property index*.

Return Value

`CSSValue` [p.146]	Returns the value of the property if it has been explicitly set for this declaration block. Returns `null` if the property has not been set.

No Exceptions

`getPropertyPriority`

Used to retrieve the priority of a CSS property (e.g. the `"important"` qualifier) if the property has been explicitly set in this declaration block.

Parameters

`propertyName` of type `DOMString` [p.21]

The name of the CSS property. See the *CSS property index*.

Return Value

`DOMString` [p.21]	A string representing the priority (e.g. `"important"`) if one exists. The empty string if none exists.

No Exceptions

`getPropertyValue`

Used to retrieve the value of a CSS property if it has been explicitly set within this declaration block.

Parameters

`propertyName` of type `DOMString` [p.21]

The name of the CSS property. See the *CSS property index*.

Return Value

`DOMString` [p.21]	Returns the value of the property if it has been explicitly set for this declaration block. Returns the empty string if the property has not been set.

No Exceptions

`item`

Used to retrieve the properties that have been explicitly set in this declaration block. The order of the properties retrieved using this method does not have to be the order in which they were set. This method can be used to iterate over all properties in this declaration block.

Parameters

`index` of type `unsigned long`

Index of the property name to retrieve.

Return Value

`DOMString` [p.21]	The name of the property at this ordinal position. The empty string if no property exists at this position.

No Exceptions

`removeProperty`

Used to remove a CSS property if it has been explicitly set within this declaration block.

Parameters

`propertyName` of type `DOMString` [p.21]

The name of the CSS property. See the *CSS property index*.

Return Value

`DOMString` [p.21]	Returns the value of the property if it has been explicitly set for this declaration block. Returns the empty string if the property has not been set or the property name does not correspond to a known CSS property.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly or the property is readonly.

`setProperty`

Used to set a property value and priority within this declaration block.

Parameters

`propertyName` of type `DOMString` [p.21]

The name of the CSS property. See the *CSS property index*.

`value` of type `DOMString`

The new value of the property.

`priority` of type `DOMString`
The new priority of the property (e.g. `"important"`).

Exceptions

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly or the property is readonly.

No Return Value

Interface *CSSValue* (introduced in **DOM Level 2**)

The `CSSValue` interface represents a simple or a complex value. A `CSSValue` object only occurs in a context of a CSS property.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSValue {
  // UnitTypes
  const unsigned short      CSS_INHERIT                    = 0;
  const unsigned short      CSS_PRIMITIVE_VALUE            = 1;
  const unsigned short      CSS_VALUE_LIST                 = 2;
  const unsigned short      CSS_CUSTOM                     = 3;

           attribute DOMString        cssText;
                                        // raises(DOMException) on setting

  readonly attribute unsigned short   valueType;
};
```

Definition group *UnitTypes*

An integer indicating which type of unit applies to the value.

Defined Constants

`CSS_CUSTOM`
The value is a custom value.

`CSS_INHERIT`
The value is inherited.

`CSS_PRIMITIVE_VALUE`
The value is a primitive value and an instance of the `CSSPrimitiveValue` [p.147] interface can be obtained by using binding-specific casting methods on this instance of the `CSSValue` interface.

`CSS_VALUE_LIST`
The value is a `CSSValue` list and an instance of the `CSSValueList` [p.154] interface can be obtained by using binding-specific casting methods on this instance of the `CSSValue` interface.

Attributes

`cssText` of type `DOMString` [p.21]
A string representation of the current value.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error (according to the attached property) or is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this value is readonly.

`valueType` of type `unsigned short`, readonly
A code defining the type of the value as defined above.

Interface *CSSPrimitiveValue* (introduced in **DOM Level 2**)

The `CSSPrimitiveValue` interface represents a single *CSS value*. This interface may be used to determine the value of a specific style property currently set in a block or to set a specific style property explicitly within the block. An instance of this interface might be obtained from the `getPropertyCSSValue` method of the `CSSStyleDeclaration` [p.142] interface. A `CSSPrimitiveValue` object only occurs in a context of a CSS property.

Conversions are allowed between absolute values (from millimeters to centimeters, from degrees to radians, and so on) but not between relative values. (For example, a pixel value cannot be converted to a centimeter value.) Percentage values can’t be converted since they are relative to the parent value (or another property value). There is one exception for color percentage values: since a color percentage value is relative to the range 0-255, a color percentage value can be converted to a number; (see also the `RGBColor` [p.154] interface).
IDL Definition

```
// Introduced in DOM Level 2:
interface CSSPrimitiveValue : CSSValue {
  // UnitTypes
  const unsigned short      CSS_UNKNOWN                    = 0;
  const unsigned short      CSS_NUMBER                     = 1;
  const unsigned short      CSS_PERCENTAGE                 = 2;
  const unsigned short      CSS_EMS                        = 3;
  const unsigned short      CSS_EXS                        = 4;
  const unsigned short      CSS_PX                         = 5;
  const unsigned short      CSS_CM                         = 6;
  const unsigned short      CSS_MM                         = 7;
  const unsigned short      CSS_IN                         = 8;
  const unsigned short      CSS_PT                         = 9;
  const unsigned short      CSS_PC                         = 10;
  const unsigned short      CSS_DEG                        = 11;
  const unsigned short      CSS_RAD                        = 12;
  const unsigned short      CSS_GRAD                       = 13;
  const unsigned short      CSS_MS                         = 14;
  const unsigned short      CSS_S                          = 15;
  const unsigned short      CSS_HZ                         = 16;
  const unsigned short      CSS_KHZ                        = 17;
  const unsigned short      CSS_DIMENSION                  = 18;
  const unsigned short      CSS_STRING                     = 19;
  const unsigned short      CSS_URI                        = 20;
  const unsigned short      CSS_IDENT                      = 21;
  const unsigned short      CSS_ATTR                       = 22;
  const unsigned short      CSS_COUNTER                    = 23;
  const unsigned short      CSS_RECT                       = 24;
  const unsigned short      CSS_RGBCOLOR                   = 25;

  readonly attribute unsigned short   primitiveType;
  void               setFloatValue(in unsigned short unitType,
                                   in float floatValue)
                                        raises(DOMException);
  float              getFloatValue(in unsigned short unitType)
                                        raises(DOMException);
  void               setStringValue(in unsigned short stringType,
                                    in DOMString stringValue)
                                        raises(DOMException);
  DOMString          getStringValue()
                                        raises(DOMException);
  Counter            getCounterValue()
                                        raises(DOMException);
  Rect               getRectValue()
                                        raises(DOMException);
  RGBColor           getRGBColorValue()
                                        raises(DOMException);
};
```

Definition group *UnitTypes*

An integer indicating which type of unit applies to the value.

Defined Constants

`CSS_ATTR`
: The value is a *attribute function*. The value can be obtained by using the `getStringValue` method.

`CSS_CM`
: The value is a *length (cm)*. The value can be obtained by using the `getFloatValue` method.

`CSS_COUNTER`
: The value is a *counter or counters function*. The value can be obtained by using the `getCounterValue` method.

`CSS_DEG`
: The value is an *angle (deg)*. The value can be obtained by using the `getFloatValue` method.

`CSS_DIMENSION`
: The value is a number with an unknown dimension. The value can be obtained by using the `getFloatValue` method.

`CSS_EMS`
: The value is a *length (ems)*. The value can be obtained by using the `getFloatValue` method.

`CSS_EXS`
: The value is a *length (exs)*. The value can be obtained by using the `getFloatValue` method.

`CSS_GRAD`
: The value is an *angle (grad)*. The value can be obtained by using the `getFloatValue` method.

`CSS_HZ`
: The value is a *frequency (Hz)*. The value can be obtained by using the `getFloatValue` method.

`CSS_IDENT`
: The value is an *identifier*. The value can be obtained by using the `getStringValue` method.

`CSS_IN`
: The value is a *length (in)*. The value can be obtained by using the `getFloatValue` method.

`CSS_KHZ`
: The value is a *frequency (kHz)*. The value can be obtained by using the `getFloatValue` method.

`CSS_MM`
: The value is a *length (mm)*. The value can be obtained by using the `getFloatValue` method.

`CSS_MS`
: The value is a *time (ms)*. The value can be obtained by using the `getFloatValue` method.

`CSS_NUMBER`
: The value is a simple *number*. The value can be obtained by using the `getFloatValue` method.

`CSS_PC`
: The value is a *length (pc)*. The value can be obtained by using the `getFloatValue` method.

`CSS_PERCENTAGE`
: The value is a *percentage*. The value can be obtained by using the `getFloatValue` method.

`CSS_PT`
: The value is a *length (pt)*. The value can be obtained by using the `getFloatValue` method.

`CSS_PX`
: The value is a *length (px)*. The value can be obtained by using the `getFloatValue` method.

`CSS_RAD`
: The value is an *angle (rad)*. The value can be obtained by using the `getFloatValue` method.

`CSS_RECT`
: The value is a *rect function*. The value can be obtained by using the `getRectValue` method.

`CSS_RGBCOLOR`
: The value is a *RGB color*. The value can be obtained by using the `getRGBColorValue` method.

`CSS_S`
: The value is a *time (s)*. The value can be obtained by using the `getFloatValue` method.

`CSS_STRING`
: The value is a *STRING*. The value can be obtained by using the `getStringValue` method.

`CSS_UNKNOWN`
: The value is not a recognized CSS2 value. The value can only be obtained by using the `cssText` attribute.

`CSS_URI`
: The value is a *URI*. The value can be obtained by using the `getStringValue` method.

Attributes

`primitiveType` of type `unsigned short`, readonly
: The type of the value as defined by the constants specified above.

Methods

`getCounterValue`

This method is used to get the Counter value. If this CSS value doesn't contain a counter value, a `DOMException` [p.24] is raised. Modification to the corresponding style property can be achieved using the `Counter` [p.155] interface.

Return Value

`Counter` [p.155]	The Counter value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a Counter value (e.g. this is not `CSS_COUNTER`).

No Parameters

`getFloatValue`

This method is used to get a float value in a specified unit. If this CSS value doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised.

Parameters

`unitType` of type `unsigned short`

A unit code to get the float value. The unit code can only be a float unit type (i.e. `CSS_NUMBER`, `CSS_PERCENTAGE`, `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC`, `CSS_DEG`, `CSS_RAD`, `CSS_GRAD`, `CSS_MS`, `CSS_S`, `CSS_HZ`, `CSS_KHZ`, `CSS_DIMENSION`).

Return Value

`float`	The float value in the specified unit.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a float value or if the float value can't be converted into the specified unit.

`getRGBColorValue`

This method is used to get the RGB color. If this CSS value doesn't contain a RGB color value, a `DOMException` [p.24] is raised. Modification to the corresponding style property can be achieved using the `RGBColor` [p.154] interface.

Return Value

`RGBColor` [p.154]	the RGB color value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the attached property can't return a RGB color value (e.g. this is not `CSS_RGBCOLOR`).

No Parameters

`getRectValue`

This method is used to get the Rect value. If this CSS value doesn't contain a rect value, a `DOMException` [p.24] is raised. Modification to the corresponding style property can be achieved using the `Rect` [p.155] interface.

Return Value

`Rect` [p.155]	The Rect value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a Rect value. (e.g. this is not `CSS_RECT`).

No Parameters

`getStringValue`

This method is used to get the string value. If the CSS value doesn't contain a string value, a `DOMException` [p.24] is raised.

Note: Some properties (like 'font-family' or 'voice-family') convert a whitespace separated list of idents to a string.

Return Value

`DOMString` [p.21]	The string value in the current unit. The current `primitiveType` can only be a string unit type (i.e. `CSS_STRING`, `CSS_URI`, `CSS_IDENT` and `CSS_ATTR`).

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a string value.

No Parameters

`setFloatValue`

A method to set the float value with a specified unit. If the property attached with this value can not accept the specified unit or the float value, the value will be unchanged and a `DOMException` [p.24] will be raised.

Parameters

`unitType` of type `unsigned short`

A unit code as defined above. The unit code can only be a float unit type (i.e. `CSS_NUMBER`, `CSS_PERCENTAGE`, `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC`, `CSS_DEG`, `CSS_RAD`, `CSS_GRAD`, `CSS_MS`, `CSS_S`, `CSS_HZ`, `CSS_KHZ`, `CSS_DIMENSION`).

`floatValue` of type `float`

The new float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the attached property doesn't support the float value or the unit type. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setStringValue`

A method to set the string value with the specified unit. If the property attached to this value can't accept the specified unit or the string value, the value will be unchanged and a `DOMException` [p.24] will be raised.

Parameters

`stringType` of type `unsigned short`

A string code as defined above. The string code can only be a string unit type (i.e. `CSS_STRING`, `CSS_URI`, `CSS_IDENT`, and `CSS_ATTR`).

`stringValue` of type `DOMString` [p.21]

The new string value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the CSS value doesn't contain a string value or if the string value can't be converted into the specified unit. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

Interface *CSSValueList* (introduced in **DOM Level 2**)

The `CSSValueList` interface provides the abstraction of an ordered collection of CSS values.

Some properties allow an empty list into their syntax. In that case, these properties take the `none` identifier. So, an empty list means that the property has the value `none`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSSValueList : CSSValue {
  readonly attribute unsigned long    length;
  CSSValue           item(in unsigned long index);
};
```

Attributes

`length` of type `unsigned long`, readonly
: The number of `CSSValues` [p.146] in the list. The range of valid values of the indices is `0` to `length-1` inclusive.

Methods

`item`
: Used to retrieve a CSS rule by ordinal index. The order in this collection represents the order of the values in the CSS style property.

Parameters

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

Return Value

`CSSValue` [p.146]	The style rule at the `index` position in the `CSSValueList`, or `null` if that is not a valid index.

No Exceptions

Interface *RGBColor* (introduced in **DOM Level 2**)

The `RGBColor` interface is used to represent any *RGB color* value. This interface reflects the values in the underlying style property. Hence, modifications made to the `CSSPrimitiveValue` [p.147] objects modify the style property.

A specified RGB color is not clipped (even if the number is outside the range 0-255 or 0%-100%). A computed RGB color is clipped depending on the device.

Even if a style sheet can only contain an integer for a color value, the internal storage of this integer is a float, and this can be used as a float in the specified or the computed style.

A color percentage value can always be converted to a number and vice versa.

IDL Definition

```
// Introduced in DOM Level 2:
interface RGBColor {
  readonly attribute CSSPrimitiveValue  red;
  readonly attribute CSSPrimitiveValue  green;
  readonly attribute CSSPrimitiveValue  blue;
};
```

Attributes

`blue` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the blue value of the RGB color.

`green` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the green value of the RGB color.

`red` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the red value of the RGB color.

Interface *Rect* (introduced in **DOM Level 2**)

The `Rect` interface is used to represent any *rect* value. This interface reflects the values in the underlying style property. Hence, modifications made to the `CSSPrimitiveValue` [p.147] objects modify the style property.

IDL Definition

```
// Introduced in DOM Level 2:
interface Rect {
  readonly attribute CSSPrimitiveValue  top;
  readonly attribute CSSPrimitiveValue  right;
  readonly attribute CSSPrimitiveValue  bottom;
  readonly attribute CSSPrimitiveValue  left;
};
```

Attributes

`bottom` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the bottom of the rect.

`left` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the left of the rect.

`right` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the right of the rect.

`top` of type `CSSPrimitiveValue` [p.147] , readonly
This attribute is used for the top of the rect.

Interface *Counter* (introduced in **DOM Level 2**)

The `Counter` interface is used to represent any *counter or counters function* value. This interface reflects the values in the underlying style property. Hence, modifications made to the `DOMString` [p.21] objects modify the style property.

IDL Definition

```
// Introduced in DOM Level 2:
interface Counter {
  readonly attribute DOMString        identifier;
  readonly attribute DOMString        listStyle;
  readonly attribute DOMString        separator;
};
```

Attributes

`identifier` of type `DOMString` [p.21] , readonly
This attribute is used for the identifier of the counter.

`listStyle` of type `DOMString` [p.21] , readonly
This attribute is used for the style of the list.

`separator` of type `DOMString` [p.21] , readonly
This attribute is used for the separator of the nested counters.

5.2.1. Override and computed style sheet

Interface *ViewCSS* (introduced in **DOM Level 2**)

This interface represents a CSS view. The `getComputedStyle` method provides a **read only access** to the *computed values* of an element.

The expectation is that an instance of the `ViewCSS` interface can be obtained by using binding-specific casting methods on an instance of the `View` interface.

Since a computed style is related to an `Element` [p.57] node, if this element is removed from the document, the associated `CSSStyleDeclaration` [p.142] and `CSSValue` [p.146] related to this declaration are no longer valid.

IDL Definition

```
// Introduced in DOM Level 2:
interface ViewCSS : views::AbstractView {
  CSSStyleDeclaration getComputedStyle(in Element elt,
                                       in DOMString pseudoElt);
};
```

Methods

`getComputedStyle`
This method is used to get the computed style as it is defined in [CSS2].
Parameters
`elt` of type `Element` [p.57]
The element whose style is to be computed. This parameter cannot be null.

`pseudoElt` of type `DOMString` [p.21]
The pseudo-element or `null` if none.

Return Value

`CSSStyleDeclaration` [p.142]	The computed style. The `CSSStyleDeclaration` is read-only and contains only absolute values.

No Exceptions

Interface *DocumentCSS* (introduced in **DOM Level 2**)

This interface represents a document with a CSS view.

The `getOverrideStyle` method provides a mechanism through which a DOM author could effect immediate change to the style of an element without modifying the explicitly linked style sheets of a document or the inline style of elements in the style sheets. This style sheet comes after the author style sheet in the cascade algorithm and is called *override style sheet*. The override style sheet takes precedence over author style sheets. An "!important" declaration still takes precedence over a normal declaration. Override, author, and user style sheets all may contain "!important" declarations. User "!important" rules take precedence over both override and author "!important" rules, and override "!important" rules take precedence over author "!important" rules.

The expectation is that an instance of the `DocumentCSS` interface can be obtained by using binding-specific casting methods on an instance of the `Document` [p.29] interface.
IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentCSS : stylesheets::DocumentStyle {
  CSSStyleDeclaration getOverrideStyle(in Element elt,
                                       in DOMString pseudoElt);
};
```

Methods
`getOverrideStyle`
This method is used to retrieve the override style declaration for a specified element and a specified pseudo-element.
Parameters
`elt` of type `Element` [p.57]
The element whose style is to be modified. This parameter cannot be null.

`pseudoElt` of type `DOMString` [p.21]
The pseudo-element or `null` if none.

Return Value

`CSSStyleDeclaration` [p.142]	The override style declaration.

No Exceptions

5.2.2. Style sheet creation

Interface *DOMImplementationCSS* (introduced in **DOM Level 2**)

This interface allows the DOM user to create a `CSSStyleSheet` [p.134] outside the context of a document. There is no way to associate the new `CSSStyleSheet` with a document in DOM Level 2.

IDL Definition

```
// Introduced in DOM   Level 2:
interface DOMImplementationCSS : DOMImplementation {
  CSSStyleSheet      createCSSStyleSheet(in DOMString title, 
                                         in DOMString media);
};
```

Methods

`createCSSStyleSheet`

Creates a new `CSSStyleSheet` [p.134] .

Parameters

`title` of type `DOMString` [p.21]

The advisory title. See also the Style Sheet Interfaces [p.128] section.

`media` of type `DOMString`

The comma-separated list of media associated with the new style sheet. See also the Style Sheet Interfaces [p.128] section.

Return Value

`CSSStyleSheet` [p.134]	A new CSS style sheet.

No Exceptions

5.2.3. Element with CSS inline style

Interface *ElementCSSInlineStyle* (introduced in **DOM Level 2**)

Inline style information attached to elements is exposed through the `style` attribute. This represents the contents of the *STYLE* attribute for HTML elements (or elements in other schemas or DTDs which use the STYLE attribute in the same way). The expectation is that an instance of the ElementCSSInlineStyle interface can be obtained by using binding-specific casting methods on an instance of the Element interface when the element supports inline CSS style informations.

IDL Definition

```
// Introduced in DOM Level 2:
interface ElementCSSInlineStyle {
  readonly attribute CSSStyleDeclaration  style;
};
```

Attributes

`style` of type `CSSStyleDeclaration` [p.142] , readonly

The style attribute.

5.3. CSS Extended Interfaces

The interfaces found within this section are not mandatory. A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether they are supported or not. The feature string for all the extended interfaces listed in this section is "CSS2".

The following table specifies the type of `CSSValue` [p.146] used to represent each property that can be specified in a `CSSStyleDeclaration` [p.142] found in a `CSSStyleRule` [p.138] for a CSS Level 2 style sheet. The expectation is that the `CSSValue` returned from the `getPropertyCSSValue` method on the `CSSStyleDeclaration` interface can be cast down, using binding-specific casting methods, to the specific derived interface.

For properties that are represented by a custom interface (the `valueType` of the `CSSValue` [p.146] is `CSS_CUSTOM`), the name of the derived interface is specified in the table. For properties that consist of lists of values (the `valueType` of the `CSSValue` is `CSS_VALUE_LIST`), the derived interface is `CSSValueList` [p.154] . For all other properties (the `valueType` of the `CSSValue` is `CSS_PRIMITIVE_VALUE`), the derived interface is `CSSPrimitiveValue` [p.147] .

Property Name	**Representation**
azimuth	`CSS2Azimuth` [p.163]
background	`null`
background-attachment	ident
background-color	rgbcolor, ident
background-image	uri, ident
background-position	`CSS2BackgroundPosition` [p.165]
background-repeat	ident
border	`null`
border-collapse	ident
border-color	`null`
border-spacing	`CSS2BorderSpacing` [p.168]
border-style	`null`
border-top, border-right, border-bottom, border-left	`null`

border-top-color, border-right-color, border-bottom-color, border-left-color	rgbcolor, ident
border-top-style, border-right-style, border-bottom-style, border-left-style	ident
border-top-width, border-right-width, border-bottom-width, border-left-width	length, ident
border-width	`null`
bottom	length, percentage, ident
caption-side	ident
clear	ident
clip	rect, ident
color	rgbcolor, ident
content	list of string, uri, counter, attr, ident
counter-increment	list of `CSS2CounterIncrement` [p.171]
counter-reset	list of `CSS2CounterReset` [p.170]
cue	`null`
cue-after, cue-before	uri, ident
cursor	`CSS2Cursor` [p.172]
direction	ident
display	ident
elevation	angle, ident
empty-cells	ident
float	ident
font	`null`
font-family	list of strings and idents
font-size	ident, length, percentage
font-size-adjust	number, ident
font-stretch	ident
font-style	ident

font-variant	ident
font-weight	ident
height	length, percentage, ident
left	length, percentage, ident
letter-spacing	ident, length
line-height	ident, length, percentage, number
list-style	`null`
list-style-image	uri, ident
list-style-position	ident
list-style-type	ident
margin	`null`
margin-top, margin-right, margin-bottom, margin-left	length, percentage, ident
marker-offset	length, ident
max-height	length, percentage, ident
max-width	length, percentage, ident
min-height	length, percentage, ident
min-width	length, percentage, ident
orphans	number
outline	`null`
outline-color	rgbcolor, ident
outline-style	ident
outline-width	length, ident
overflow	ident
padding	`null`
padding-top, padding-right, padding-bottom, padding-left	length, percentage
page	ident
page-break-after	ident
page-break-before	ident

page-break-inside	ident
pause	`null`
pause-after, pause-before	time, percentage
pitch	frequency, identifier
pitch-range	number
play-during	`CSS2PlayDuring` [p.173]
position	ident
quotes	list of string or ident
richness	number
right	length, percentage, ident
speak	ident
speak-header	ident
speak-numeral	ident
speak-punctuation	ident
speech-rate	number, ident
stress	number
table-layout	ident
text-align	ident, string
text-decoration	list of ident
text-indent	length, percentage
text-shadow	list of `CSS2TextShadow` [p.174]
text-transform	ident
top	length, percentage, ident
unicode-bidi	ident
vertical-align	ident, percentage, length
visibility	ident
voice-family	list of strings and idents
volume	number, percentage, ident

white-space	ident
widows	number
width	length, percentage, ident
word-spacing	length, ident
z-index	ident, number

Interface *CSS2Azimuth* (introduced in **DOM Level 2**)

The `CSS2Azimuth` interface represents the *azimuth* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2Azimuth : CSSValue {
  readonly attribute unsigned short   azimuthType;
  readonly attribute DOMString        identifier;
  readonly attribute boolean          behind;
  void               setAngleValue(in unsigned short uType, 
                                   in float fValue)
                                        raises(DOMException);
  float              getAngleValue(in unsigned short uType)
                                        raises(DOMException);
  void               setIdentifier(in DOMString ident, 
                                   in boolean b)
                                        raises(DOMException);
};
```

Attributes

`azimuthType` of type `unsigned short`, readonly
A code defining the type of the value as defined in `CSSValue` [p.146] . It would be one of `CSS_DEG`, `CSS_RAD`, `CSS_GRAD` or `CSS_IDENT`.

`behind` of type `boolean`, readonly
`behind` indicates whether the behind identifier has been set.

`identifier` of type `DOMString` [p.21] , readonly
If `azimuthType` is `CSS_IDENT`, `identifier` contains one of left-side, far-left, left, center-left, center, center-right, right, far-right, right-side, leftwards, rightwards. The empty string if none is set.

Methods

`getAngleValue`
Used to retrieved the float value of the azimuth property.
Parameters

`uType` of type `unsigned short`
The unit type can be only an angle unit type (`CSS_DEG`, `CSS_RAD` or `CSS_GRAD`).

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the unit type is invalid.

`setAngleValue`
A method to set the angle value with a specified unit. This method will unset any previously set identifier value.
Parameters
`uType` of type `unsigned short`
The unitType could only be one of `CSS_DEG`, `CSS_RAD` or `CSS_GRAD`).

`fValue` of type `float`
The new float value of the angle.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the unit type is invalid. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setIdentifier`
Setting the identifier for the azimuth property will unset any previously set angle value. The value of `azimuthType` is set to `CSS_IDENT`
Parameters
`ident` of type `DOMString` [p.21]
The new identifier. If the identifier is "leftwards" or "rightward", the behind attribute is ignored.

`b` of type `boolean`
The new value for behind.

Exceptions

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified `identifier` has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

Interface *CSS2BackgroundPosition* (introduced in **DOM Level 2**)

The `CSS2BackgroundPosition` interface represents the *background-position* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2BackgroundPosition : CSSValue {
  readonly attribute unsigned short   horizontalType;
  readonly attribute unsigned short   verticalType;
  readonly attribute DOMString        horizontalIdentifier;
  readonly attribute DOMString        verticalIdentifier;
  float              getHorizontalPosition(in float hType)
                                        raises(DOMException);
  float              getVerticalPosition(in float vType)
                                        raises(DOMException);
  void               setHorizontalPosition(in unsigned short hType, 
                                           in float value)
                                        raises(DOMException);
  void               setVerticalPosition(in unsigned short vType, 
                                         in float value)
                                        raises(DOMException);
  void               setPositionIdentifier(in DOMString hIdentifier, 
                                           in DOMString vIdentifier)
                                        raises(DOMException);
};
```

Attributes

`horizontalIdentifier` of type `DOMString` [p.21] , readonly

If `horizontalType` is `CSS_IDENT` or `CSS_INHERIT`, this attribute contains the string representation of the ident, otherwise it contains an empty string.

`horizontalType` of type `unsigned short`, readonly

A code defining the type of the horizontal value. It would be one of `CSS_PERCENTAGE`, `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC` or `CSS_IDENT`. If one of horizontal or vertical is `CSS_IDENT`, it’s guaranteed that the other is the same.

`verticalIdentifier` of type `DOMString` [p.21] , readonly
If `verticalType` is `CSS_IDENT` or `CSS_INHERIT`, this attribute contains the string representation of the ident, otherwise it contains an empty string. The value is `"center"` if only the horizontalIdentifier has been set.

`verticalType` of type `unsigned short`, readonly
A code defining the type of the horizontal value. The code can be one of the following units : `CSS_PERCENTAGE`, `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC`, `CSS_IDENT`, `CSS_INHERIT`. If one of horizontal or vertical is `CSS_IDENT` or `CSS_INHERIT`, it's guaranteed that the other is the same.

Methods

`getHorizontalPosition`
This method is used to get the float value in a specified unit if the `horizontalPosition` represents a length or a percentage. If the float doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised.

Parameters

`hType` of type `float`
The horizontal unit.

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the property doesn't contain a float or the value can't be converted.

`getVerticalPosition`
This method is used to get the float value in a specified unit if the `verticalPosition` represents a length or a percentage. If the float doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised. The value is `50%` if only the horizontal value has been specified.

Parameters

`vType` of type `float`
The vertical unit.

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the property doesn't contain a float or the value can't be converted.

`setHorizontalPosition`

This method is used to set the horizontal position with a specified unit. If the vertical value is not a percentage or a length, it sets the vertical position to `50%`.

Parameters

`hType` of type `unsigned short`

The specified unit (a length or a percentage).

`value` of type `float`

The new value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the specified unit is not a length or a percentage.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setPositionIdentifier`

Sets the identifiers. If the second identifier is the empty string, the vertical identifier is set to its default value (`"center"`).

Parameters

`hIdentifier` of type `DOMString` [p.21]

The new horizontal identifier.

`vIdentifier` of type `DOMString`

The new vertical identifier.

Exceptions

`DOMException` [p.24]	SYNTAX_ERR: Raised if the identifiers have a syntax error and are unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setVerticalPosition`

This method is used to set the vertical position with a specified unit. If the horizontal value is not a percentage or a length, it sets the vertical position to `50%`.

Parameters

`vType` of type `unsigned short`
The specified unit (a length or a percentage).

`value` of type `float`
The new value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the specified unit is not a length or a percentage. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

Interface *CSS2BorderSpacing* (introduced in **DOM Level 2**)

The `CSS2BorderSpacing` interface represents the *border-spacing* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2BorderSpacing : CSSValue {
  readonly attribute unsigned short   horizontalType;
  readonly attribute unsigned short   verticalType;
  float              getHorizontalSpacing(in float hType)
                                        raises(DOMException);
  float              getVerticalSpacing(in float vType)
                                        raises(DOMException);
  void               setHorizontalSpacing(in unsigned short hType, 
                                          in float value)
                                        raises(DOMException);
  void               setVerticalSpacing(in unsigned short vType, 
                                        in float value)
                                        raises(DOMException);
};
```

Attributes

`horizontalType` of type `unsigned short`, readonly
The A code defining the type of the value as defined in `CSSValue` [p.146] . It would be one of `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT` or `CSS_PC`.

`verticalType` of type `unsigned short`, readonly
The A code defining the type of the value as defined in `CSSValue` [p.146] . It would be one of `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC`

or `CSS_INHERIT`.

Methods

`getHorizontalSpacing`

This method is used to get the float value in a specified unit if the `horizontalSpacing` represents a length. If the float doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised.

Parameters

`hType` of type `float`

The horizontal unit.

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the property doesn't contain a float or the value can't be converted.

`getVerticalSpacing`

This method is used to get the float value in a specified unit if the `verticalSpacing` represents a length. If the float doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised. The value is `0` if only the horizontal value has been specified.

Parameters

`vType` of type `float`

The vertical unit.

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the property doesn't contain a float or the value can't be converted.

`setHorizontalSpacing`

This method is used to set the horizontal spacing with a specified unit. If the vertical value is a length, it sets the vertical spacing to `0`.

Parameters

`hType` of type `unsigned short`

The horizontal unit.

`value` of type `float`
The new value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the specified unit is not a length. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setVerticalSpacing`
This method is used to set the vertical spacing with a specified unit. If the horizontal value is not a length, it sets the vertical spacing to `0`.
Parameters
`vType` of type `unsigned short`
The vertical unit.

`value` of type `float`
The new value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the specified unit is not a length or a percentage. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

Interface *CSS2CounterReset* (introduced in **DOM Level 2**)

The `CSS2CounterReset` interface represents a simple value for the *counter-reset* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.
IDL Definition

```
// Introduced in DOM Level   2:
interface CSS2CounterReset : CSSValue {
           attribute DOMString        identifier;
                                        // raises(DOMException) on setting

           attribute short            reset;
                                        // raises(DOMException) on setting

};
```

Attributes

`identifier` of type `DOMString` [p.21]

The element name.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified identifier has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this identifier is readonly.

`reset` of type `short`

The reset (default value is 0).

Exceptions on setting

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this identifier is readonly.

Interface *CSS2CounterIncrement* (introduced in **DOM Level 2**)

The `CSS2CounterIncrement` interface represents a simple value for the *counter-increment* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM   Level 2:
interface CSS2CounterIncrement : CSSValue {
           attribute DOMString        identifier;
                                        // raises(DOMException) on setting

           attribute short            increment;
                                        // raises(DOMException) on setting

};
```

Attributes

`identifier` of type `DOMString` [p.21]
The element name.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified identifier has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this identifier is readonly.

`increment` of type `short`
The increment. (Default value is 1.)
Exceptions on setting

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this identifier is readonly.

Interface *CSS2Cursor* (introduced in **DOM Level 2**)

The `CSS2Cursor` interface represents the *cursor* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2Cursor : CSSValue {
  readonly attribute CSSValueList     uris;
           attribute DOMString        predefinedCursor;
                                        // raises(DOMException) on setting

};
```

Attributes
`predefinedCursor` of type `DOMString` [p.21]
This identifier represents a generic cursor name or an empty string.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

`uris` of type `CSSValueList` [p.154] , readonly
`uris` represents the list of URIs (`CSS_URI`) on the cursor property. The list can be empty.

Interface *CSS2PlayDuring* (introduced in **DOM Level 2**)

The `CSS2PlayDuring` interface represents the *play-during* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2PlayDuring : CSSValue {
  readonly attribute unsigned short   playDuringType;
           attribute DOMString        playDuringIdentifier;
                                        // raises(DOMException) on setting

           attribute DOMString        uri;
                                        // raises(DOMException) on setting

           attribute boolean          mix;
                                        // raises(DOMException) on setting

           attribute boolean          repeat;
                                        // raises(DOMException) on setting

};
```

Attributes

`mix` of type `boolean`

`true` if the sound should be mixed. It will be ignored if the attribute doesn't contain a `uri`.

Exceptions on setting

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

`playDuringIdentifier` of type `DOMString` [p.21]

One of `"inherit"`, `"auto"`, `"none"` or the empty string if the `playDuringType` is `CSS_UNKNOWN`. On setting, it will set the `uri` to the empty string and `mix` and `repeat` to `false`.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

`playDuringType` of type `unsigned short`, readonly

A code defining the type of the value as defined in `CSSvalue`. It would be one of `CSS_UNKNOWN` or `CSS_IDENT`.

`repeat` of type `boolean`

`true` if the sound should be repeated. It will be ignored if the attribute doesn't contain a `uri`.

Exceptions on setting

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

`uri` of type `DOMString` [p.21]

The sound specified by the `uri`. It will set the `playDuringType` attribute to `CSS_UNKNOWN`.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

Interface *CSS2TextShadow* (introduced in **DOM Level 2**)

The `CSS2TextShadow` interface represents a simple value for the *text-shadow* CSS Level 2 property.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2TextShadow : CSSValue {
  readonly attribute CSSValue         color;
  readonly attribute CSSValue         horizontal;
  readonly attribute CSSValue         vertical;
  readonly attribute CSSValue         blur;
};
```

Attributes

`blur` of type `CSSValue` [p.146] , readonly

The blur radius of the text shadow. `0` if no length has been specified.

`color` of type `CSSValue` [p.146] , readonly

Specifies the color of the text shadow. The CSS Value can contain an empty string if no color has been specified.

`horizontal` of type `CSSValue` [p.146] , readonly

The horizontal position of the text shadow. `0` if no length has been specified.

`vertical` of type `CSSValue` [p.146] , readonly
: The vertical position of the text shadow. `0` if no length has been specified.

The following table specifies the type of `CSSValue` [p.146] used to represent each descriptor that can be specified in a `CSSStyleDeclaration` [p.142] found in a `CSSFontFaceRule` [p.140] for a CSS Level 2 style sheet.

Property Name	**Representation**
font-family	list of strings and idents
font-style	list of idents
font-variant	list of idents
font-weight	list of idents
font-stretch	list of idents
font-size	list of lengths or ident
unicode-range	list of strings. Example: "U+370-3FF", "U+1F??"
units-per-em	number
src	list of `CSS2FontFaceSrc` [p.176]
panose-1	list of integers
stemv	number
stemh	number
slope	number
cap-height	number
x-height	number
ascent	number
descent	number
widths	list of `CSS2FontFaceWidths` [p.177]
bbox	list of numbers
definition-src	uri
baseline	number
centerline	number
mathline	number
topline	number

Interface *CSS2FontFaceSrc* (introduced in **DOM Level 2**)

The `CSS2FontFaceSrc` interface represents the *src* CSS Level 2 descriptor.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.
IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2FontFaceSrc : CSSValue {
           attribute DOMString        uri;
                                        // raises(DOMException) on setting

  readonly attribute CSSValueList     format;
           attribute DOMString        fontFaceName;
                                        // raises(DOMException) on setting

};
```

Attributes

`fontFaceName` of type `DOMString` [p.21]
Specifies the full font name of a locally installed font.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

`format` of type `CSSValueList` [p.154] , readonly
This attribute contains a list of strings for the format CSS function.

`uri` of type `DOMString` [p.21]
Specifies the source of the font, empty string otherwise.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

Interface *CSS2FontFaceWidths* (introduced in **DOM Level 2**)

The `CSS2FontFaceWidths` interface represents a simple value for the *widths* CSS Level 2 descriptor.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2FontFaceWidths : CSSValue {
           attribute DOMString        urange;
                                        // raises(DOMException) on setting

  readonly attribute CSSValueList     numbers;
};
```

Attributes

`numbers` of type `CSSValueList` [p.154] , readonly
A list of numbers representing the glyph widths.

`urange` of type `DOMString` [p.21]
The range for the characters.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this declaration is readonly.

The following table specifies the type of `CSSValue` [p.146] used to represent each property that can be specified in a `CSSStyleDeclaration` [p.142] found in a `CSSPageRule` [p.140] for a CSS Level 2 style sheet.

Property Name	**Representation**
margin	`null`
margin-top, margin-right, margin-bottom, margin-left	length (no CSS_EMS and CSS_EXS), percentage, ident
marks	list of idents
size	`CSS2PageSize` [p.178]

Interface *CSS2PageSize* (introduced in **DOM Level 2**)

The `CSS2PageSize` interface represents the *size* CSS Level 2 descriptor.

For this extension of the `CSSValue` [p.146] interface, the `valueType` attribute of the underlying `CSSValue` interface shall be `CSS_CUSTOM`.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2PageSize : CSSValue {
  readonly attribute unsigned short   widthType;
  readonly attribute unsigned short   heightType;
  readonly attribute DOMString        identifier;
  float              getWidth(in float wType)
                                        raises(DOMException);
  float              getHeightSize(in float hType)
                                        raises(DOMException);
  void               setWidthSize(in unsigned short wType, 
                                  in float value)
                                        raises(DOMException);
  void               setHeightSize(in unsigned short hType, 
                                   in float value)
                                        raises(DOMException);
  void               setIdentifier(in DOMString ident)
                                        raises(DOMException);
};
```

Attributes

`heightType` of type `unsigned short`, readonly
A code defining the type of the height of the page. It would be one of `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC` or `CSS_IDENT`. If one of width or height is `CSS_IDENT`, it's guaranteed that the other is the same.

`identifier` of type `DOMString` [p.21] , readonly
If `width` is `CSS_IDENT`, this attribute contains the string representation of the ident, otherwise it contains an empty string.

`widthType` of type `unsigned short`, readonly
A code defining the type of the width of the page. It would be one of `CSS_EMS`, `CSS_EXS`, `CSS_PX`, `CSS_CM`, `CSS_MM`, `CSS_IN`, `CSS_PT`, `CSS_PC` or `CSS_IDENT`.

Methods

`getHeightSize`
This method is used to get the float value in a specified unit if the `heightType` represents a length. If the float doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised. If only the width value has been specified, the height value is the same.

Parameters

`hType` of type `float`
The height unit.

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the property doesn't contain a float or the value can't be converted.

`getWidth`

This method is used to get the float value in a specified unit if the `widthType` represents a length. If the float doesn't contain a float value or can't be converted into the specified unit, a `DOMException` [p.24] is raised.

Parameters

`wType` of type `float`

The width unit.

Return Value

`float`	The float value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the property doesn't contain a float or the value can't be converted.

`setHeightSize`

This method is used to set the height position with a specified unit. If the `widthType` is not a length, it sets the width position to the same value.

Parameters

`hType` of type `unsigned short`

The height unit.

`value` of type `float`

The new value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the specified unit is not a length or a percentage. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setIdentifier`

Sets the identifier.

Parameters

`ident` of type `DOMString` [p.21]
The new identifier.

Exceptions

`DOMException` [p.24]	SYNTAX_ERR: Raised if the identifier has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

`setWidthSize`
This method is used to set the width position with a specified unit. If the `heightType` is not a length, it sets the height position to the same value.
Parameters
`wType` of type `unsigned short`
The width unit.

`value` of type `float`
The new value.

Exceptions

`DOMException` [p.24]	INVALID_ACCESS_ERR: Raised if the specified unit is not a length or a percentage. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

No Return Value

Interface *CSS2Properties* (introduced in **DOM Level 2**)

The `CSS2Properties` interface represents a convenience mechanism for retrieving and setting properties within a `CSSStyleDeclaration` [p.142] . The attributes of this interface correspond to all the *properties specified in CSS2*. Getting an attribute of this interface is equivalent to calling the `getPropertyValue` method of the `CSSStyleDeclaration` interface. Setting an attribute of this interface is equivalent to calling the `setProperty` method of the `CSSStyleDeclaration` interface.

A compliant implementation is not required to implement the `CSS2Properties` interface. If an implementation does implement this interface, the expectation is that language-specific methods can be used to cast from an instance of the `CSSStyleDeclaration` [p.142] interface to the `CSS2Properties` interface.

If an implementation does implement this interface, it is expected to understand the specific syntax of the shorthand properties, and apply their semantics; when the `margin` property is set, for example, the `marginTop`, `marginRight`, `marginBottom` and `marginLeft` properties are actually being set by the underlying implementation.

When dealing with CSS "shorthand" properties, the shorthand properties should be decomposed into their component longhand properties as appropriate, and when querying for their value, the form returned should be the shortest form exactly equivalent to the declarations made in the ruleset. However, if there is no shorthand declaration that could be added to the ruleset without changing in any way the rules already declared in the ruleset (i.e., by adding longhand rules that were previously not declared in the ruleset), then the empty string should be returned for the shorthand property.

For example, querying for the `font` property should not return "normal normal normal 14pt/normal Arial, sans-serif", when "14pt Arial, sans-serif" suffices. (The normals are initial values, and are implied by use of the longhand property.)

If the values for all the longhand properties that compose a particular string are the initial values, then a string consisting of all the initial values should be returned (e.g. a `border-width` value of "medium" should be returned as such, not as "").

For some shorthand properties that take missing values from other sides, such as the `margin`, `padding`, and `border-[width|style|color]` properties, the minimum number of sides possible should be used; i.e., "0px 10px" will be returned instead of "0px 10px 0px 10px".

If the value of a shorthand property can not be decomposed into its component longhand properties, as is the case for the `font` property with a value of "menu", querying for the values of the component longhand properties should return the empty string.

IDL Definition

```
// Introduced in DOM Level 2:
interface CSS2Properties {
           attribute DOMString        azimuth;
                                        // raises(DOMException) on setting

           attribute DOMString        background;
                                        // raises(DOMException) on setting

           attribute DOMString        backgroundAttachment;
                                        // raises(DOMException) on setting

           attribute DOMString        backgroundColor;
                                        // raises(DOMException) on setting

           attribute DOMString        backgroundImage;
                                        // raises(DOMException) on setting

           attribute DOMString        backgroundPosition;
                                        // raises(DOMException) on setting

           attribute DOMString        backgroundRepeat;
                                        // raises(DOMException) on setting
```

```
attribute DOMString        border;
                             // raises(DOMException) on setting

attribute DOMString        borderCollapse;
                             // raises(DOMException) on setting

attribute DOMString        borderColor;
                             // raises(DOMException) on setting

attribute DOMString        borderSpacing;
                             // raises(DOMException) on setting

attribute DOMString        borderStyle;
                             // raises(DOMException) on setting

attribute DOMString        borderTop;
                             // raises(DOMException) on setting

attribute DOMString        borderRight;
                             // raises(DOMException) on setting

attribute DOMString        borderBottom;
                             // raises(DOMException) on setting

attribute DOMString        borderLeft;
                             // raises(DOMException) on setting

attribute DOMString        borderTopColor;
                             // raises(DOMException) on setting

attribute DOMString        borderRightColor;
                             // raises(DOMException) on setting

attribute DOMString        borderBottomColor;
                             // raises(DOMException) on setting

attribute DOMString        borderLeftColor;
                             // raises(DOMException) on setting

attribute DOMString        borderTopStyle;
                             // raises(DOMException) on setting

attribute DOMString        borderRightStyle;
                             // raises(DOMException) on setting

attribute DOMString        borderBottomStyle;
                             // raises(DOMException) on setting

attribute DOMString        borderLeftStyle;
                             // raises(DOMException) on setting

attribute DOMString        borderTopWidth;
                             // raises(DOMException) on setting

attribute DOMString        borderRightWidth;
                             // raises(DOMException) on setting
```

```
attribute DOMString        borderBottomWidth;
                             // raises(DOMException) on setting

attribute DOMString        borderLeftWidth;
                             // raises(DOMException) on setting

attribute DOMString        borderWidth;
                             // raises(DOMException) on setting

attribute DOMString        bottom;
                             // raises(DOMException) on setting

attribute DOMString        captionSide;
                             // raises(DOMException) on setting

attribute DOMString        clear;
                             // raises(DOMException) on setting

attribute DOMString        clip;
                             // raises(DOMException) on setting

attribute DOMString        color;
                             // raises(DOMException) on setting

attribute DOMString        content;
                             // raises(DOMException) on setting

attribute DOMString        counterIncrement;
                             // raises(DOMException) on setting

attribute DOMString        counterReset;
                             // raises(DOMException) on setting

attribute DOMString        cue;
                             // raises(DOMException) on setting

attribute DOMString        cueAfter;
                             // raises(DOMException) on setting

attribute DOMString        cueBefore;
                             // raises(DOMException) on setting

attribute DOMString        cursor;
                             // raises(DOMException) on setting

attribute DOMString        direction;
                             // raises(DOMException) on setting

attribute DOMString        display;
                             // raises(DOMException) on setting

attribute DOMString        elevation;
                             // raises(DOMException) on setting

attribute DOMString        emptyCells;
                             // raises(DOMException) on setting
```

```
attribute DOMString        cssFloat;
                             // raises(DOMException) on setting

attribute DOMString        font;
                             // raises(DOMException) on setting

attribute DOMString        fontFamily;
                             // raises(DOMException) on setting

attribute DOMString        fontSize;
                             // raises(DOMException) on setting

attribute DOMString        fontSizeAdjust;
                             // raises(DOMException) on setting

attribute DOMString        fontStretch;
                             // raises(DOMException) on setting

attribute DOMString        fontStyle;
                             // raises(DOMException) on setting

attribute DOMString        fontVariant;
                             // raises(DOMException) on setting

attribute DOMString        fontWeight;
                             // raises(DOMException) on setting

attribute DOMString        height;
                             // raises(DOMException) on setting

attribute DOMString        left;
                             // raises(DOMException) on setting

attribute DOMString        letterSpacing;
                             // raises(DOMException) on setting

attribute DOMString        lineHeight;
                             // raises(DOMException) on setting

attribute DOMString        listStyle;
                             // raises(DOMException) on setting

attribute DOMString        listStyleImage;
                             // raises(DOMException) on setting

attribute DOMString        listStylePosition;
                             // raises(DOMException) on setting

attribute DOMString        listStyleType;
                             // raises(DOMException) on setting

attribute DOMString        margin;
                             // raises(DOMException) on setting

attribute DOMString        marginTop;
                             // raises(DOMException) on setting
```

```
  attribute DOMString        marginRight;
                               // raises(DOMException) on setting

  attribute DOMString        marginBottom;
                               // raises(DOMException) on setting

  attribute DOMString        marginLeft;
                               // raises(DOMException) on setting

  attribute DOMString        markerOffset;
                               // raises(DOMException) on setting

  attribute DOMString        marks;
                               // raises(DOMException) on setting

  attribute DOMString        maxHeight;
                               // raises(DOMException) on setting

  attribute DOMString        maxWidth;
                               // raises(DOMException) on setting

  attribute DOMString        minHeight;
                               // raises(DOMException) on setting

  attribute DOMString        minWidth;
                               // raises(DOMException) on setting

  attribute DOMString        orphans;
                               // raises(DOMException) on setting

  attribute DOMString        outline;
                               // raises(DOMException) on setting

  attribute DOMString        outlineColor;
                               // raises(DOMException) on setting

  attribute DOMString        outlineStyle;
                               // raises(DOMException) on setting

  attribute DOMString        outlineWidth;
                               // raises(DOMException) on setting

  attribute DOMString        overflow;
                               // raises(DOMException) on setting

  attribute DOMString        padding;
                               // raises(DOMException) on setting

  attribute DOMString        paddingTop;
                               // raises(DOMException) on setting

  attribute DOMString        paddingRight;
                               // raises(DOMException) on setting

  attribute DOMString        paddingBottom;
                               // raises(DOMException) on setting
```

```
attribute DOMString        paddingLeft;
                             // raises(DOMException) on setting

attribute DOMString        page;
                             // raises(DOMException) on setting

attribute DOMString        pageBreakAfter;
                             // raises(DOMException) on setting

attribute DOMString        pageBreakBefore;
                             // raises(DOMException) on setting

attribute DOMString        pageBreakInside;
                             // raises(DOMException) on setting

attribute DOMString        pause;
                             // raises(DOMException) on setting

attribute DOMString        pauseAfter;
                             // raises(DOMException) on setting

attribute DOMString        pauseBefore;
                             // raises(DOMException) on setting

attribute DOMString        pitch;
                             // raises(DOMException) on setting

attribute DOMString        pitchRange;
                             // raises(DOMException) on setting

attribute DOMString        playDuring;
                             // raises(DOMException) on setting

attribute DOMString        position;
                             // raises(DOMException) on setting

attribute DOMString        quotes;
                             // raises(DOMException) on setting

attribute DOMString        richness;
                             // raises(DOMException) on setting

attribute DOMString        right;
                             // raises(DOMException) on setting

attribute DOMString        size;
                             // raises(DOMException) on setting

attribute DOMString        speak;
                             // raises(DOMException) on setting

attribute DOMString        speakHeader;
                             // raises(DOMException) on setting

attribute DOMString        speakNumeral;
                             // raises(DOMException) on setting
```

```
  attribute DOMString        speakPunctuation;
                               // raises(DOMException) on setting

  attribute DOMString        speechRate;
                               // raises(DOMException) on setting

  attribute DOMString        stress;
                               // raises(DOMException) on setting

  attribute DOMString        tableLayout;
                               // raises(DOMException) on setting

  attribute DOMString        textAlign;
                               // raises(DOMException) on setting

  attribute DOMString        textDecoration;
                               // raises(DOMException) on setting

  attribute DOMString        textIndent;
                               // raises(DOMException) on setting

  attribute DOMString        textShadow;
                               // raises(DOMException) on setting

  attribute DOMString        textTransform;
                               // raises(DOMException) on setting

  attribute DOMString        top;
                               // raises(DOMException) on setting

  attribute DOMString        unicodeBidi;
                               // raises(DOMException) on setting

  attribute DOMString        verticalAlign;
                               // raises(DOMException) on setting

  attribute DOMString        visibility;
                               // raises(DOMException) on setting

  attribute DOMString        voiceFamily;
                               // raises(DOMException) on setting

  attribute DOMString        volume;
                               // raises(DOMException) on setting

  attribute DOMString        whiteSpace;
                               // raises(DOMException) on setting

  attribute DOMString        widows;
                               // raises(DOMException) on setting

  attribute DOMString        width;
                               // raises(DOMException) on setting

  attribute DOMString        wordSpacing;
                               // raises(DOMException) on setting
```

```
           attribute DOMString        zIndex;
                                        // raises(DOMException) on setting

};
```

Attributes

`azimuth` of type `DOMString` [p.21]

See the *azimuth property definition* in CSS2.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`background` of type `DOMString` [p.21]

See the *background property definition* in CSS2.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`backgroundAttachment` of type `DOMString` [p.21]

See the *background-attachment property definition* in CSS2.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`backgroundColor` of type `DOMString` [p.21]

See the *background-color property definition* in CSS2.

Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`backgroundImage` of type `DOMString` [p.21]
See the *background-image property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`backgroundPosition` of type `DOMString` [p.21]
See the *background-position property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`backgroundRepeat` of type `DOMString` [p.21]
See the *background-repeat property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`border` of type `DOMString` [p.21]
See the *border property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderBottom` of type `DOMString` [p.21]
See the *border-bottom property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderBottomColor` of type `DOMString` [p.21]
See the *border-bottom-color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderBottomStyle` of type `DOMString` [p.21]
See the *border-bottom-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderBottomWidth` of type `DOMString` [p.21]
See the *border-bottom-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderCollapse` of type `DOMString` [p.21]
See the *border-collapse property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderColor` of type `DOMString` [p.21]
See the *border-color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderLeft` of type `DOMString` [p.21]
See the *border-left property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderLeftColor` of type `DOMString` [p.21]
See the *border-left-color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderLeftStyle` of type `DOMString` [p.21]
See the *border-left-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderLeftWidth` of type `DOMString` [p.21]
See the *border-left-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderRight` of type `DOMString` [p.21]
See the *border-right property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderRightColor` of type `DOMString` [p.21]
See the *border-right-color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderRightStyle` of type `DOMString` [p.21]
See the *border-right-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderRightWidth` of type `DOMString` [p.21]
See the *border-right-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderSpacing` of type `DOMString` [p.21]
See the *border-spacing property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderStyle` of type `DOMString` [p.21]
See the *border-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderTop` of type `DOMString` [p.21]
See the *border-top property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderTopColor` of type `DOMString` [p.21]
See the *border-top-color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderTopStyle` of type `DOMString` [p.21]
See the *border-top-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderTopWidth` of type `DOMString` [p.21]
See the *border-top-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`borderWidth` of type `DOMString` [p.21]
See the *border-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`bottom` of type `DOMString` [p.21]
See the *bottom property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`captionSide` of type `DOMString` [p.21]
See the *caption-side property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`clear` of type `DOMString` [p.21]
See the *clear property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`clip` of type `DOMString` [p.21]
See the *clip property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`color` of type `DOMString` [p.21]
See the *color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`content` of type `DOMString` [p.21]
See the *content property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`counterIncrement` of type `DOMString` [p.21]
See the *counter-increment property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`counterReset` of type `DOMString` [p.21]
See the *counter-reset property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`cssFloat` of type `DOMString` [p.21]
See the *float property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`cue` of type `DOMString` [p.21]
See the *cue property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`cueAfter` of type `DOMString` [p.21]
See the *cue-after property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`cueBefore` of type `DOMString` [p.21]
See the *cue-before property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`cursor` of type `DOMString` [p.21]
See the *cursor property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`direction` of type `DOMString` [p.21]
See the *direction property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`display` of type `DOMString` [p.21]
See the *display property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`elevation` of type `DOMString` [p.21]
See the *elevation property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`emptyCells` of type `DOMString` [p.21]
See the *empty-cells property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`font` of type `DOMString` [p.21]
See the *font property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontFamily` of type `DOMString` [p.21]
See the *font-family property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontSize` of type `DOMString` [p.21]
See the *font-size property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontSizeAdjust` of type `DOMString` [p.21]
See the *font-size-adjust property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontStretch` of type `DOMString` [p.21]
See the *font-stretch property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontStyle` of type `DOMString` [p.21]
See the *font-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontVariant` of type `DOMString` [p.21]
See the *font-variant property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`fontWeight` of type `DOMString` [p.21]
See the *font-weight property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`height` of type `DOMString` [p.21]
See the *height property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`left` of type `DOMString` [p.21]
See the *left property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`letterSpacing` of type `DOMString` [p.21]
See the *letter-spacing property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`lineHeight` of type `DOMString` [p.21]
See the *line-height property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`listStyle` of type `DOMString` [p.21]
See the *list-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`listStyleImage` of type `DOMString` [p.21]
See the *list-style-image property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`listStylePosition` of type `DOMString` [p.21]
See the *list-style-position property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`listStyleType` of type `DOMString` [p.21]
See the *list-style-type property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`margin` of type `DOMString` [p.21]
See the *margin property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`marginBottom` of type `DOMString` [p.21]
See the *margin-bottom property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`marginLeft` of type `DOMString` [p.21]
See the *margin-left property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`marginRight` of type `DOMString` [p.21]
See the *margin-right property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`marginTop` of type `DOMString` [p.21]
See the *margin-top property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`markerOffset` of type `DOMString` [p.21]
See the *marker-offset property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`marks` of type `DOMString` [p.21]
See the *marks property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`maxHeight` of type `DOMString` [p.21]
See the *max-height property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`maxWidth` of type `DOMString` [p.21]
See the *max-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`minHeight` of type `DOMString` [p.21]
See the *min-height property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`minWidth` of type `DOMString` [p.21]
See the *min-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`orphans` of type `DOMString` [p.21]
See the *orphans property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`outline` of type `DOMString` [p.21]
See the *outline property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`outlineColor` of type `DOMString` [p.21]
See the *outline-color property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`outlineStyle` of type `DOMString` [p.21]
See the *outline-style property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`outlineWidth` of type `DOMString` [p.21]
See the *outline-width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`overflow` of type `DOMString` [p.21]
See the *overflow property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`padding` of type `DOMString` [p.21]
See the *padding property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`paddingBottom` of type `DOMString` [p.21]
See the *padding-bottom property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`paddingLeft` of type `DOMString` [p.21]
See the *padding-left property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`paddingRight` of type `DOMString` [p.21]
See the *padding-right property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`paddingTop` of type `DOMString` [p.21]
See the *padding-top property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`page` of type `DOMString` [p.21]
See the *page property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pageBreakAfter` of type `DOMString` [p.21]
See the *page-break-after property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pageBreakBefore` of type `DOMString` [p.21]
See the *page-break-before property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pageBreakInside` of type `DOMString` [p.21]
See the *page-break-inside property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pause` of type `DOMString` [p.21]
See the *pause property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pauseAfter` of type `DOMString` [p.21]
See the *pause-after property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pauseBefore` of type `DOMString` [p.21]
See the *pause-before property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pitch` of type `DOMString` [p.21]
See the *pitch property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`pitchRange` of type `DOMString` [p.21]
See the *pitch-range property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`playDuring` of type `DOMString` [p.21]
See the *play-during property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`position` of type `DOMString` [p.21]
See the *position property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`quotes` of type `DOMString` [p.21]
See the *quotes property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`richness` of type `DOMString` [p.21]
See the *richness property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`right` of type `DOMString` [p.21]
See the *right property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`size` of type `DOMString` [p.21]
See the *size property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`speak` of type `DOMString` [p.21]
See the *speak property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable.
	NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`speakHeader` of type `DOMString` [p.21]
See the *speak-header property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`speakNumeral` of type `DOMString` [p.21]
See the *speak-numeral property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`speakPunctuation` of type `DOMString` [p.21]
See the *speak-punctuation property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`speechRate` of type `DOMString` [p.21]
See the *speech-rate property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`stress` of type `DOMString` [p.21]
See the *stress property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`tableLayout` of type `DOMString` [p.21]
See the *table-layout property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`textAlign` of type `DOMString` [p.21]
See the *text-align property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`textDecoration` of type `DOMString` [p.21]
See the *text-decoration property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`textIndent` of type `DOMString` [p.21]
See the *text-indent property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`textShadow` of type `DOMString` [p.21]
See the *text-shadow property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`textTransform` of type `DOMString` [p.21]
See the *text-transform property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`top` of type `DOMString` [p.21]
See the *top property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`unicodeBidi` of type `DOMString` [p.21]
See the *unicode-bidi property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`verticalAlign` of type `DOMString` [p.21]
See the *vertical-align property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`visibility` of type `DOMString` [p.21]
See the *visibility property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`voiceFamily` of type `DOMString` [p.21]
See the *voice-family property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`volume` of type `DOMString` [p.21]
See the *volume property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`whiteSpace` of type `DOMString` [p.21]
See the *white-space property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`widows` of type `DOMString` [p.21]
See the *widows property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`width` of type `DOMString` [p.21]
See the *width property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`wordSpacing` of type `DOMString` [p.21]
See the *word-spacing property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

`zIndex` of type `DOMString` [p.21]
See the *z-index property definition* in CSS2.
Exceptions on setting

`DOMException` [p.24]	SYNTAX_ERR: Raised if the new value has a syntax error and is unparsable. NO_MODIFICATION_ALLOWED_ERR: Raised if this property is readonly.

6. Document Object Model Events

Editors

Tom Pixley, Netscape Communications Corporation

6.1. Overview of the DOM Level 2 Event Model

The DOM Level 2 Event Model is designed with two main goals. The first goal is the design of a generic event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event. Additionally, the specification will provide standard sets of events for user interface control and document mutation notifications, including defined contextual information for each of these event sets.

The second goal of the event model is to provide a common subset of the current event systems used in *DOM Level 0* [p.448] browsers. This is intended to foster interoperability of existing scripts and content. It is not expected that this goal will be met with full backwards compatibility. However, the specification attempts to achieve this when possible.

The following sections of the Event Model specification define both the specification for the DOM Event Model and a number of compliant event sets designed for use within the model. The Event Model consists of the two sections on event propagation and event listener registration and the Event interface. A DOM consumer can use the `hasFeature` of the `DOMImplementation` [p.26] interface to determine whether the Event Model has been implemented by a DOM implementation. The feature string for the Event Model is "Events". The existence within an implementation of each of the individual event sets can also be queried using the `hasFeature` method. Each event set describes its own feature string in the event set listing.

6.1.1. Terminology

UI events
User interface events. These events are generated by user interaction through an external device (mouse, keyboard, etc.)

UI Logical events
Device independent user interface events such as focus change messages or element triggering notifications.

Mutation events
Events caused by any action which modifies the structure of the document.

Capturing
The process by which an event can be handled by one of the event’s target’s ancestors before being handled by the event’s target.

Bubbling
The process by which an event propagates upward through its ancestors after being handled by the event’s target.

Cancelable
A designation for events which indicates that upon handling the event the client may choose to prevent the DOM implementation from processing any default action associated with the event.

6.2. Description of event flow

Event flow is the process through which the an event originates from the DOM implementation and is passed into the Document Object Model. The methods of event capture and event bubbling, along with various event listener registration techniques, allow the event to then be handled in a number of ways. It can be handled locally at the `EventTarget` level or centrally from an `EventTarget` [p.224] higher in the document tree.

6.2.1. Basic event flow

Each event has an `EventTarget` [p.224] toward which the event is directed by the DOM implementation. This `EventTarget` is specified in the `Event` [p.227] 's `target` attribute. When the event reaches the target, any event listeners registered on the `EventTarget` are triggered. Although all `EventListeners` [p.226] on the `EventTarget` are guaranteed to be triggered by any event which is received by that `EventTarget`, no specification is made as to the order in which they will receive the event with regards to the other `EventListeners` [p.226] on the `EventTarget`. If neither event capture or event bubbling are in use for that particular event, the event flow process will complete after all listeners have been triggered. If event capture or event bubbling is in use, the event flow will be modified as described in the sections below.

Any exceptions thrown inside an `EventListener` [p.226] will not stop propagation of the event. It will continue processing any additional `EventListener` in the described manner.

It is expected that actions taken by `EventListener` [p.226] s may cause additional events to fire. Additional events should be handled in a synchronous manner and may cause reentrancy into the event model.

6.2.2. Event capture

Event capture is the process by which an EventListener registered on an ancestor of the event's target can intercept events of a given type before they are received by the event's target. Capture operates from the top of the tree, generally the `Document` [p.29] , downward, making it the symmetrical opposite of bubbling which is described below. The chain of `EventTarget` [p.224] s from the top of the tree to the event's target is determined before the initial dispatch of the event. If modifications occur to the tree during event processing, event flow will proceed based on the initial state of the tree.

An `EventListener` [p.226] being registered on an `EventTarget` [p.224] may choose to have that `EventListener` capture events by specifying the `useCapture` parameter of the `addEventListener` method to be `true`. Thereafter, when an event of the given type is dispatched toward a descendant of the capturing object, the event will trigger any capturing event listeners of the appropriate type which exist in the direct line between the top of the document and the event's target. This downward propagation continues until the event's target is reached. A capturing `EventListener` will not be triggered by events dispatched directly to the `EventTarget` upon which it is registered.

If the capturing `EventListener` [p.226] wishes to prevent further processing of the event from occurring it may call the `stopProgagation` method of the `Event` [p.227] interface. This will prevent further dispatch of the event, although additional `EventListeners` registered at the same hierarchy level will still receive the event. Once an event's `stopPropagation` method has been called, further calls to that method have no additional effect. If no additional capturers exist and `stopPropagation` has not been called, the event triggers the appropriate `EventListeners` on the target itself.

Although event capture is similar to the delegation based event model in which all interested parties register their listeners directly on the target about which they wish to receive notifications, it is different in two important respects. First, event capture only allows interception of events which are targeted at descendants of the capturing `EventTarget` [p.224] . It does not allow interception of events targeted to the capturer's ancestors, its siblings, or its sibling's descendants. Secondly, event capture is not specified for a single `EventTarget`, it is specified for a specific type of event. Once specified, event capture intercepts all events of the specified type targeted toward any of the capturer's descendants.

6.2.3. Event bubbling

Events which are designated as bubbling will initially proceed with the same event flow as non-bubbling events. The event is dispatched to its target `EventTarget` [p.224] and any event listeners found there are triggered. Bubbling events will then trigger any additional event listeners found by following the `EventTarget`'s parent chain upward, checking for any event listeners registered on each successive `EventTarget`. This upward propagation will continue up to and including the `Document` [p.29] . `EventListener` [p.226] s registered as capturers will not be triggered during this phase. The chain of `EventTargets` from the event target to the top of the tree is determined before the initial dispatch of the event. If modifications occur to the tree during event processing, event flow will proceed based on the initial state of the tree.

Any event handler may choose to prevent further event propagation by calling the `stopPropagation` method of the `Event` [p.227] interface. If any `EventListener` [p.226] calls this method, all additional `EventListeners` on the current `EventTarget` [p.224] will be triggered but bubbling will cease at that level. Only one call to `stopPropagation` is required to prevent further bubbling.

6.2.4. Event cancelation

Some events are specified as cancelable. For these events, the DOM implementation generally has a default action associated with the event. An example of this is a hyperlink in a web browser. When the user clicks on the hyperlink the default action is generally to active that hyperlink. Before processing these events, the implementation must check for event listeners registered to receive the event and dispatch the event to those listeners. These listeners then have the option of canceling the implementation's default action or allowing the default action to proceed. In the case of the hyperlink in the browser, canceling the action would have the result of not activating the hyperlink.

Cancelation is accomplished by calling the `Event` [p.227] 's `preventDefault` method. If one or more `EventListeners` [p.226] call `preventDefault` during any phase of event flow the default action will be canceled.

Different implementations will specify their own default actions, if any, associated with each event. The DOM does not attempt to specify these actions.

6.3. Event listener registration

6.3.1. Event registration interfaces

Interface *EventTarget* (introduced in **DOM Level 2**)

The `EventTarget` interface is implemented by all `Nodes` [p.38] in an implementation which supports the DOM Event Model. The interface allows registration and removal of `EventListeners` [p.226] on an `EventTarget` and dispatch of events to that `EventTarget`.

IDL Definition

```
// Introduced in DOM Level 2:
interface EventTarget {
  void               addEventListener(in DOMString type, 
                                      in EventListener listener, 
                                      in boolean useCapture);
  void               removeEventListener(in DOMString type, 
                                         in EventListener listener, 
                                         in boolean useCapture);
  boolean            dispatchEvent(in Event evt)
                                        raises(EventException);
};
```

Methods

`addEventListener`

This method allows the registration of event listeners on the event target. If an `EventListener` [p.226] is added to an `EventTarget` while it is processing an event, it will not be triggered by the current actions but may be triggered during a later stage of event flow, such as the bubbling phase.

If multiple identical `EventListener` [p.226] s are registered on the same `EventTarget` with the same parameters the duplicate instances are discarded. They do not cause the `EventListener` to be called twice and since they are discarded they do not need to be removed with the `removeEventListener` method.

Parameters

`type` of type `DOMString` [p.21]

The event type for which the user is registering

`listener` of type `EventListener` [p.226]

The `listener` parameter takes an interface implemented by the user which contains the methods to be called when the event occurs.

`useCapture` of type `boolean`

If true, `useCapture` indicates that the user wishes to initiate capture. After initiating capture, all events of the specified type will be dispatched to the registered `EventListener` before being dispatched to any `EventTargets` beneath them in the tree. Events which are bubbling upward through the tree will not trigger an

`EventListener` designated to use capture.

No Return Value
No Exceptions

`dispatchEvent`

This method allows the dispatch of events into the implementations event model. Events dispatched in this manner will have the same capturing and bubbling behavior as events dispatched directly by the implementation. The target of the event is the `EventTarget` on which `dispatchEvent` is called.

Parameters

`evt` of type `Event` [p.227]

Specifies the event type, behavior, and contextual information to be used in processing the event.

Return Value

`boolean`	The return value of `dispatchEvent` indicates whether any of the listeners which handled the event called `preventDefault`. If `preventDefault` was called the value is false, else the value is true.

Exceptions

`EventException` [p.229]	UNSPECIFIED_EVENT_TYPE_ERR: Raised if the `Event` [p.227] 's type was not specified by initializing the event before `dispatchEvent` was called. Specification of the `Event`'s type as `null` or an empty string will also trigger this exception.

`removeEventListener`

This method allows the removal of event listeners from the event target. If an `EventListener` [p.226] is removed from an `EventTarget` while it is processing an event, it will not be triggered by the current actions. `EventListener`s can never be invoked after being removed.

Calling `removeEventListener` with arguments which do not identify any currently registered `EventListener` [p.226] on the `EventTarget` has no effect.

Parameters

`type` of type `DOMString` [p.21]

Specifies the event type of the `EventListener` [p.226] being removed.

`listener` of type `EventListener` [p.226]

The `EventListener` parameter indicates the `EventListener` to be removed.

`useCapture` of type `boolean`
: Specifies whether the `EventListener` being removed was registered as a capturing listener or not. If a listener was registered twice, one with capture and one without, each must be removed separately. Removal of a capturing listener does not affect a non-capturing version of the same listener, and vice versa.

No Return Value
No Exceptions

Interface *EventListener* (introduced in **DOM Level 2**)

The `EventListener` interface is the primary method for handling events. Users implement the `EventListener` interface and register their listener on an `EventTarget` [p.224] using the `AddEventListener` method. The users should also remove their `EventListener` from its `EventTarget` after they have completed using the listener.

When a `Node` [p.38] is copied using the `cloneNode` method the `EventListener`s attached to the source `Node` are not attached to the copied `Node`. If the user wishes the same `EventListener`s to be added to the newly created copy the user must add them manually.
IDL Definition

```
// Introduced in DOM Level 2:
interface EventListener {
  void               handleEvent(in Event evt);
};
```

Methods
`handleEvent`
: This method is called whenever an event occurs of the type for which the `EventListener` interface was registered.
Parameters
`evt` of type `Event` [p.227]
: The `Event` contains contextual information about the event. It also contains the `stopPropagation` and `preventDefault` methods which are used in determining the event’s flow and default action.

No Return Value
No Exceptions

6.3.2. Interaction with HTML 4.0 event listeners

In HTML 4.0, event listeners were specified as attributes of an element. As such, registration of a second event listener of the same type would replace the first listener. The DOM Event Model allows registration of multiple event listeners on a single `EventTarget` [p.224] . To achieve this, event listeners are no longer stored as attribute values.

In order to achieve compatibility with HTML 4.0, implementors may view the setting of attributes which represent event handlers as the creation and registration of an `EventListener` on the `EventTarget` [p.224] . The value of `useCapture` defaults to `false`. This `EventListener` [p.226] behaves in the same manner as any other `EventListeners` which may be registered on the `EventTarget`. If the attribute representing the event listener is changed, this may be viewed as the removal of the previously registered `EventListener` and the registration of a new one. No technique is provided to allow HTML 4.0 event listeners access to the context information defined for each event.

6.4. Event interface

Interface *Event* (introduced in **DOM Level 2**)

The `Event` interface is used to provide contextual information about an event to the handler processing the event. An object which implements the `Event` interface is generally passed as the first parameter to an event handler. More specific context information is passed to event handlers by deriving additional interfaces from `Event` which contain information directly relating to the type of event they accompany. These derived interfaces are also implemented by the object passed to the event listener.

IDL Definition

```
// Introduced in DOM Level 2:
interface Event {
  // PhaseType
  const unsigned short      CAPTURING_PHASE                = 1;
  const unsigned short      AT_TARGET                      = 2;
  const unsigned short      BUBBLING_PHASE                 = 3;

  readonly attribute DOMString        type;
  readonly attribute EventTarget      target;
  readonly attribute EventTarget      currentTarget;
  readonly attribute unsigned short   eventPhase;
  readonly attribute boolean          bubbles;
  readonly attribute boolean          cancelable;
  readonly attribute DOMTimeStamp     timeStamp;
  void               stopPropagation();
  void               preventDefault();
  void               initEvent(in DOMString eventTypeArg,
                               in boolean canBubbleArg,
                               in boolean cancelableArg);
};
```

Definition group *PhaseType*

An integer indicating which phase of event flow is being processed.

Defined Constants

`AT_TARGET`

The event is currently being evaluated at the target `EventTarget` [p.224] .

`BUBBLING_PHASE`
The current event phase is the bubbling phase.

`CAPTURING_PHASE`
The current event phase is the capturing phase.

Attributes

`bubbles` of type `boolean`, readonly
Used to indicate whether or not an event is a bubbling event. If the event can bubble the value is true, else the value is false.

`cancelable` of type `boolean`, readonly
Used to indicate whether or not an event can have its default action prevented. If the default action can be prevented the value is true, else the value is false.

`currentTarget` of type `EventTarget` [p.224] , readonly
Used to indicate the `EventTarget` [p.224] whose `EventListeners` [p.226] are currently being processed. This is particularly useful during capturing and bubbling.

`eventPhase` of type `unsigned short`, readonly
Used to indicate which phase of event flow is currently being evaluated.

`target` of type `EventTarget` [p.224] , readonly
Used to indicate the `EventTarget` [p.224] to which the event was originally dispatched.

`timeStamp` of type `DOMTimeStamp` [p.22] , readonly
Used to specify the time (in milliseconds relative to the epoch) at which the event was created. Due to the fact that some systems may not provide this information the value of `timeStamp` may be not available for all events. When not available, a value of 0 will be returned. Examples of epoch time are the time of the system start or 0:0:0 UTC 1st January 1970.

`type` of type `DOMString` [p.21] , readonly
The name of the event (case-insensitive). The name must be an *XML name* [p.451] .

Methods

`initEvent`
The `initEvent` method is used to initialize the value of an `Event` created through the `DocumentEvent` [p.230] interface. This method may only be called before the `Event` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times the final invocation takes precedence. If called from a subclass of `Event` interface only the values specified in the `initEvent` method are modified, all other attributes are left unchanged.
Parameters
`eventTypeArg` of type `DOMString` [p.21]
Specifies the event type. This type may be any event type currently defined in this specification or a new event type.. The string must be an *XML name* [p.451] .
Any new event type must not begin with any upper, lower, or mixed case version of

the string "DOM". This prefix is reserved for future DOM event sets. It is also strongly recommended that third parties adding their own events use their own prefix to avoid confusion and lessen the probability of conflicts with other new events.

`canBubbleArg` of type `boolean`
Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`
Specifies whether or not the event's default action can be prevented.

No Return Value
No Exceptions

`preventDefault`
If an event is cancelable, the `preventDefault` method is used to signify that the event is to be canceled, meaning any default action normally taken by the implementation as a result of the event will not occur. If, during any stage of event flow, the `preventDefault` method is called the event is canceled. Any default action associated with the event will not occur. Calling this method for a non-cancelable event has no effect. Once `preventDefault` has been called it will remain in effect throughout the remainder of the event's propagation. This method may be used during any stage of event flow.
No Parameters
No Return Value
No Exceptions

`stopPropagation`
The `stopPropagation` method is used prevent further propagation of an event during event flow. If this method is called by any `EventListener` [p.226] the event will cease propagating through the tree. The event will complete dispatch to all listeners on the current `EventTarget` [p.224] before event flow stops. This method may be used during any stage of event flow.
No Parameters
No Return Value
No Exceptions

Exception *EventException* introduced in **DOM Level 2**

Event operations may throw an `EventException` [p.229] as specified in their method descriptions.
IDL Definition

```
// Introduced in DOM Level 2:
exception EventException {
  unsigned short   code;
};

// EventExceptionCode
const unsigned short      UNSPECIFIED_EVENT_TYPE_ERR     = 0;
```

Definition group *EventExceptionCode*

An integer indicating the type of error generated.

Defined Constants

`UNSPECIFIED_EVENT_TYPE_ERR`

If the `Event` [p.227] 's type was not specified by initializing the event before the method was called. Specification of the Event's type as `null` or an empty string will also trigger this exception.

6.5. DocumentEvent interface

Interface *DocumentEvent* (introduced in **DOM Level 2**)

The `DocumentEvent` interface provides a mechanism by which the user can create an Event of a type supported by the implementation. It is expected that the `DocumentEvent` interface will be implemented on the same object which implements the `Document` [p.29] interface in an implementation which supports the Event model.

IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentEvent {
  Event              createEvent(in DOMString eventType)
                                        raises(DOMException);
};
```

Methods

`createEvent`

Parameters

`eventType` of type `DOMString` [p.21]

The `eventType` parameter specifies the type of `Event` [p.227] interface to be created. If the `Event` interface specified is supported by the implementation this method will return a new `Event` of the interface type requested. If the `Event` is to be dispatched via the `dispatchEvent` method the appropriate event init method must be called after creation in order to initialize the `Event`'s values. As an example, a user wishing to synthesize some kind of `UIEvent` [p.231] would call `createEvent` with the parameter "UIEvents". The `initUIEvent` method could then be called on the newly created `UIEvent` to set the specific type of UIEvent to be dispatched and set its context information.

The `createEvent` method is used in creating `Event` [p.227] s when it is either inconvenient or unnecessary for the user to create an `Event` themselves. In cases where the implementation provided `Event` is insufficient, users may supply their own `Event` implementations for use with the `dispatchEvent` method.

Return Value

`Event` [p.227]	The newly created `Event`

Exceptions

`DOMException` [p.24]	NOT_SUPPORTED_ERR: Raised if the implementation does not support the type of `Event` [p.227] interface requested

6.6. Event set definitions

The DOM Level 2 Event Model allows a DOM implementation to support multiple sets of events. The model has been designed to allow addition of new event sets as is required. The DOM will not attempt to define all possible events. For purposes of interoperability, the DOM will define a set of user interface events including lower level device dependent events, a set of UI logical events, and a set of document mutation events. Any new event types defined by third parties must not begin with any upper, lower, or mixed case version of the string "DOM". This prefix is reserved for future DOM event sets. It is also strongly recommended that third parties adding their own events use their own prefix to avoid confusion and lessen the probability of conflicts with other new events.

6.6.1. User Interface event types

The User Interface event set is composed of events listed in HTML 4.0 and additional events which are supported in *DOM Level 0* [p.448] browsers.

A DOM consumer can use the `hasFeature` of the `DOMImplementation` [p.26] interface to determine whether the User Interface event set has been implemented by a DOM implementation. The feature string for this event set is "UIEvents". This string is also used with the `createEvent` method.

Interface *UIEvent* (introduced in **DOM Level 2**)

The `UIEvent` interface provides specific contextual information associated with User Interface events.

IDL Definition

```
// Introduced in DOM Level 2:
interface UIEvent : Event {
  readonly attribute views::AbstractView  view;
  readonly attribute long             detail;
  void               initUIEvent(in DOMString typeArg,
                                 in boolean canBubbleArg,
                                 in boolean cancelableArg,
                                 in views::AbstractView viewArg,
                                 in long detailArg);
};
```

Attributes

`detail` of type `long`, readonly

Specifies some detail information about the `Event` [p.227] , depending on the type of event.

`view` of type `views::AbstractView`, readonly
: The `view` attribute identifies the `AbstractView` [p.125] from which the event was generated.

Methods

`initUIEvent`
: The `initUIEvent` method is used to initialize the value of a `UIEvent` created through the `DocumentEvent` [p.230] interface. This method may only be called before the `UIEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

Parameters

`typeArg` of type `DOMString` [p.21]
: Specifies the event type.

`canBubbleArg` of type `boolean`
: Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`
: Specifies whether or not the event's default action can be prevented.

`viewArg` of type `views::AbstractView`
: Specifies the `Event` [p.227] 's `AbstractView` [p.125] .

`detailArg` of type `long`
: Specifies the `Event` [p.227] 's detail.

No Return Value
No Exceptions

The different types of such events that can occur are:

DOMFocusIn
: The DOMFocusIn event occurs when an `EventTarget` [p.224] receives focus, for instance via a pointing device being moved onto an element or by tabbing navigation to the element. Unlike the HTML event focus, DOMFocusIn can be applied to any focusable `EventTarget`, not just FORM controls.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

DOMFocusOut
: The DOMFocusOut event occurs when a `EventTarget` [p.224] loses focus, for instance via a pointing device being moved out of an element or by tabbing navigation out of the element. Unlike the HTML event blur, DOMFocusOut can be applied to any focusable `EventTarget`, not just FORM controls.

- Bubbles: Yes
- Cancelable: No

- Context Info: None

DOMActivate

The activate event occurs when an element is activated, for instance, thru a mouse click or a keypress. A numerical argument is provided to give an indication of the type of activation that occurs: 1 for a simple activation (e.g. a simple click or Enter), 2 for hyperactivation (for instance a double click or Shift Enter).

- Bubbles: Yes
- Cancelable: Yes
- Context Info: detail (the numerical value)

6.6.2. Mouse event types

The Mouse event set is composed of events listed in HTML 4.0 and additional events which are supported in *DOM Level 0* [p.448] browsers. This event set is specifically designed for use with mouse input devices.

A DOM consumer can use the `hasFeature` of the `DOMImplementation` [p.26] interface to determine whether the User Interface event set has been implemented by a DOM implementation. The feature string for this event set is "MouseEvents". This string is also used with the `createEvent` method.

Interface *MouseEvent* (introduced in **DOM Level 2**)

The `MouseEvent` interface provides specific contextual information associated with Mouse events.

The `detail` attribute inherited from `UIEvent` [p.231] indicates the number of times a mouse button has been pressed and released over the same screen location during a user action. The attribute value is 1 when the user begins this action and increments by 1 for each full sequence of pressing and releasing. If the user moves the mouse between the mousedown and mouseup the value will be set to 0, indicating that no click is occurring.

In the case of nested elements mouse events are always targeted at the most deeply nested element. Ancestors of the targeted element may use bubbling to obtain notification of mouse events which occur within its descendent elements.

IDL Definition

```
// Introduced in DOM Level 2:
interface MouseEvent : UIEvent {
  readonly attribute long             screenX;
  readonly attribute long             screenY;
  readonly attribute long             clientX;
  readonly attribute long             clientY;
  readonly attribute boolean          ctrlKey;
  readonly attribute boolean          shiftKey;
  readonly attribute boolean          altKey;
  readonly attribute boolean          metaKey;
  readonly attribute unsigned short   button;
  readonly attribute EventTarget      relatedTarget;
  void               initMouseEvent(in DOMString typeArg,
                                    in boolean canBubbleArg,
                                    in boolean cancelableArg,
                                    in views::AbstractView viewArg,
                                    in long detailArg,
                                    in long screenXArg,
                                    in long screenYArg,
                                    in long clientXArg,
                                    in long clientYArg,
                                    in boolean ctrlKeyArg,
                                    in boolean altKeyArg,
                                    in boolean shiftKeyArg,
                                    in boolean metaKeyArg,
                                    in unsigned short buttonArg,
                                    in EventTarget relatedTargetArg);
};
```

Attributes

`altKey` of type `boolean`, readonly
: Used to indicate whether the ’alt’ key was depressed during the firing of the event. On some platforms this key may map to an alternative key name.

`button` of type `unsigned short`, readonly
: During mouse events caused by the depression or release of a mouse button, `button` is used to indicate which mouse button changed state. The values for `button` range from zero to indicate the left button of the mouse, one to indicate the middle button if present, and two to indicate the right button. For mice configured for left handed use in which the button actions are reversed the values are instead read from right to left.

`clientX` of type `long`, readonly
: The horizontal coordinate at which the event occurred relative to the DOM implementation’s client area.

`clientY` of type `long`, readonly
: The vertical coordinate at which the event occurred relative to the DOM implementation’s client area.

`ctrlKey` of type `boolean`, readonly
: Used to indicate whether the ’ctrl’ key was depressed during the firing of the event.

`metaKey` of type `boolean`, readonly
: Used to indicate whether the ’meta’ key was depressed during the firing of the event. On some platforms this key may map to an alternative key name.

`relatedTarget` of type `EventTarget` [p.224] , readonly
: Used to identify a secondary `EventTarget` [p.224] related to a UI event. Currently this attribute is used with the mouseover event to indicate the `EventTarget` which the pointing device exited and with the mouseout event to indicate the `EventTarget` which the pointing device entered.

`screenX` of type `long`, readonly
: The horizontal coordinate at which the event occurred relative to the origin of the screen coordinate system.

`screenY` of type `long`, readonly
: The vertical coordinate at which the event occurred relative to the origin of the screen coordinate system.

`shiftKey` of type `boolean`, readonly
: Used to indicate whether the ’shift’ key was depressed during the firing of the event.

Methods

`initMouseEvent`

The `initMouseEvent` method is used to initialize the value of a `MouseEvent` created through the `DocumentEvent` [p.230] interface. This method may only be called before the `MouseEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

Parameters

`typeArg` of type `DOMString` [p.21]
: Specifies the event type.

`canBubbleArg` of type `boolean`
: Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`
: Specifies whether or not the event’s default action can be prevented.

`viewArg` of type `views::AbstractView`
: Specifies the `Event` [p.227] ’s `AbstractView` [p.125] .

`detailArg` of type `long`
: Specifies the `Event` [p.227] ’s mouse click count.

`screenXArg` of type `long`
: Specifies the `Event` [p.227] ’s screen x coordinate

`screenYArg` of type `long`
: Specifies the `Event` [p.227] ’s screen y coordinate

`clientXArg` of type `long`
: Specifies the `Event` [p.227] ’s client x coordinate

`clientYArg` of type `long`
: Specifies the `Event` [p.227] ’s client y coordinate

`ctrlKeyArg` of type `boolean`
Specifies whether or not control key was depressed during the `Event` [p.227] .

`altKeyArg` of type `boolean`
Specifies whether or not alt key was depressed during the `Event` [p.227] .

`shiftKeyArg` of type `boolean`
Specifies whether or not shift key was depressed during the `Event` [p.227] .

`metaKeyArg` of type `boolean`
Specifies whether or not meta key was depressed during the `Event` [p.227] .

`buttonArg` of type `unsigned short`
Specifies the `Event` [p.227] 's mouse button.

`relatedTargetArg` of type `EventTarget` [p.224]
Specifies the `Event` [p.227] 's related `EventTarget`.

No Return Value
No Exceptions

The different types of Mouse events that can occur are:

click
The click event occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is:

```
mousedown
mouseup
click
```

If multiple clicks occur at the same screen location, the sequence repeats with the `detail` attribute incrementing with each repetition. This event is valid for most elements.
- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, button, detail

mousedown
The mousedown event occurs when the pointing device button is pressed over an element. This event is valid for most elements.
- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, button, detail

mouseup
The mouseup event occurs when the pointing device button is released over an element. This event is valid for most elements.
- Bubbles: Yes

- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, button, detail

mouseover

The mouseover event occurs when the pointing device is moved onto an element. This event is valid for most elements.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, relatedTarget indicates the `EventTarget` [p.224] the pointing device is exiting.

mousemove

The mousemove event occurs when the pointing device is moved while it is over an element. This event is valid for most elements.

- Bubbles: Yes
- Cancelable: No
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey

mouseout

The mouseout event occurs when the pointing device is moved away from an element. This event is valid for most elements..

- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, relatedTarget indicates the `EventTarget` [p.224] the pointing device is entering.

6.6.3. Key events

The DOM Level 2 Event specification does not provide a key event set. An event set designed for use with keyboard input devices will be included in a later version of the DOM specification.

6.6.4. Mutation event types

The mutation event set is designed to allow notification of any changes to the structure of a document, including attr and text modifications. It may be noted that none of the mutation events listed are designated as cancelable. This stems from the fact that it is very difficult to make use of existing DOM interfaces which cause document modifications if any change to the document might or might not take place due to cancelation of the related event. Although this is still a desired capability, it was decided that it would be better left until the addition of transactions into the DOM.

Many single modifications of the tree can cause multiple mutation events to be fired. Rather than attempt to specify the ordering of mutation events due to every possible modification of the tree, the ordering of these events is left to the implementation.

A DOM consumer can use the `hasFeature` of the `DOMImplementation` [p.26] interface to determine whether the mutation event set has been implemented by a DOM implementation. The feature string for this event set is "MutationEvents". This string is also used with the `createEvent` method.

Interface *MutationEvent* (introduced in **DOM Level 2**)

The `MutationEvent` interface provides specific contextual information associated with Mutation events.

IDL Definition

```
// Introduced in DOM Level 2:
interface MutationEvent : Event {
  readonly attribute Node             relatedNode;
  readonly attribute DOMString        prevValue;
  readonly attribute DOMString        newValue;
  readonly attribute DOMString        attrName;
  void               initMutationEvent(in DOMString typeArg,
                                        in boolean canBubbleArg,
                                        in boolean cancelableArg,
                                        in Node relatedNodeArg,
                                        in DOMString prevValueArg,
                                        in DOMString newValueArg,
                                        in DOMString attrNameArg);
};
```

Attributes

`attrName` of type `DOMString` [p.21] , readonly

`attrName` indicates the name of the changed `Attr` [p.56] node in a DOMAttrModified event.

`newValue` of type `DOMString` [p.21] , readonly

`newValue` indicates the new value of the `Attr` [p.56] node in DOMAttrModified events, and of the `CharacterData` [p.52] node in DOMCharDataModified events.

`prevValue` of type `DOMString` [p.21] , readonly

`prevValue` indicates the previous value of the `Attr` [p.56] node in DOMAttrModified events, and of the `CharacterData` [p.52] node in DOMCharDataModified events.

`relatedNode` of type `Node` [p.38] , readonly

`relatedNode` is used to identify a secondary node related to a mutation event. For example, if a mutation event is dispatched to a node indicating that its parent has changed, the `relatedNode` is the changed parent. If an event is instead dispatch to a subtree indicating a node was changed within it, the `relatedNode` is the changed node.

Methods

`initMutationEvent`

The `initMutationEvent` method is used to initialize the value of a `MutationEvent` created through the `DocumentEvent` [p.230] interface. This method may only be called before the `MutationEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

Parameters

`typeArg` of type `DOMString` [p.21]
: Specifies the event type.

`canBubbleArg` of type `boolean`
: Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`
: Specifies whether or not the event's default action can be prevented.

`relatedNodeArg` of type `Node` [p.38]
: Specifies the `Event` [p.227] 's related Node

`prevValueArg` of type `DOMString`
: Specifies the `Event` [p.227] 's `prevValue` attribute

`newValueArg` of type `DOMString`
: Specifies the `Event` [p.227] 's `newValue` attribute

`attrNameArg` of type `DOMString`
: Specifies the `Event` [p.227] 's `attrName` attribute

No Return Value
No Exceptions

The different types of Mutation events that can occur are:

DOMSubtreeModified
This is a general event for notification of all changes to the document. It can be used instead of the more specific events listed below. It may be fired after a single modification to the document or, at the implementation's discretion, after multiple changes have occurred. The latter use should generally be used to accomodate multiple changes which occur either simultaneously or in rapid succession. The target of this event is the lowest common parent of the changes which have taken place. This event is dispatched after any other events caused by the mutation have fired.
- Bubbles: Yes
- Cancelable: No
- Context Info: None

DOMNodeInserted
Fired when a node has been added as a child of another node. This event is dispatched after the insertion has taken place. The target of this event is the node being inserted.
- Bubbles: Yes
- Cancelable: No
- Context Info: relatedNode holds the parent node

DOMNodeRemoved
Fired when a node is being removed from another node. This event is dispatched before the node is removed from the tree. The target of this event is the node being removed.
- Bubbles: Yes
- Cancelable: No

- Context Info: relatedNode holds the parent node

DOMNodeRemovedFromDocument

Fired when a node is being removed from a document, either through direct removal of the Node or removal of a subtree in which it is contained. This event is dispatched before the removal takes place. The target of this event is the Node being removed. If the Node is being directly removed the DOMNodeRemoved event will fire before the DOMNodeRemovedFromDocument event.

- Bubbles: No
- Cancelable: No
- Context Info: None

DOMNodeInsertedIntoDocument

Fired when a node is being inserted into a document, either through direct insertion of the Node or insertion of a subtree in which it is contained. This event is dispatched after the insertion has taken place. The target of this event is the node being inserted. If the Node is being directly inserted the DOMNodeInserted event will fire before the DOMNodeInsertedIntoDocument event.

- Bubbles: No
- Cancelable: No
- Context Info: None

DOMAttrModified

Fired after an `Attr` [p.56] has been modified on a node. The target of this event is the `Node` [p.38] whose `Attr` changed. The values of prevValue and newValue may be the empty string in cases where an attribute has been added or removed.

- Bubbles: Yes
- Cancelable: No
- Context Info: attrName, prevValue, newValue

DOMCharacterDataModified

Fired after CharacterData within a node has been modified but the node itself has not been inserted or deleted. This event is also triggered by modifications to PI elements. The target of this event is the CharacterData node.

- Bubbles: Yes
- Cancelable: No
- Context Info: prevValue, newValue

6.6.5. HTML event types

The HTML event set is composed of events listed in HTML 4.0 and additional events which are supported in *DOM Level 0* [p.448] browsers.

A DOM consumer can use the `hasFeature` of the `DOMImplementation` [p.26] interface to determine whether the HTML event set has been implemented by a DOM implementation. The feature string for this event set is "HTMLEvents". This string is also used with the `createEvent` method.

The HTML events use the base DOM Event interface to pass contextual information.

The different types of such events that can occur are:

load

The load event occurs when the DOM implementation finishes loading all content within a document, all frames within a FRAMESET, or an OBJECT element.

- Bubbles: No
- Cancelable: No
- Context Info: None

unload

The unload event occurs when the DOM implementation removes a document from a window or frame. This event is valid for BODY and FRAMESET elements.

- Bubbles: No
- Cancelable: No
- Context Info: None

abort

The abort event occurs when page loading is stopped before an image has been allowed to completely load. This event applies to OBJECT elements.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

error

The error event occurs when an image does not load properly or when an error occurs during script execution. This event is valid for OBJECT elements, BODY elements, and FRAMESET element.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

select

The select event occurs when a user selects some text in a text field. This event is valid for INPUT and TEXTAREA elements.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

change

The change event occurs when a control loses the input focus and its value has been modified since gaining focus. This event is valid for INPUT, SELECT, and TEXTAREA. element.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

submit

The submit event occurs when a form is submitted. This event only applies to the FORM element.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: None

reset

The reset event occurs when a form is reset. This event only applies to the FORM element.

- Bubbles: Yes
- Cancelable: No

- Context Info: None

focus

The focus event occurs when an element receives focus either via a pointing device or by tabbing navigation. This event is valid for the following elements: LABEL, INPUT, SELECT, TEXTAREA, and BUTTON.

- Bubbles: No
- Cancelable: No
- Context Info: None

blur

The blur event occurs when an element loses focus either via the pointing device or by tabbing navigation. This event is valid for the following elements: LABEL, INPUT, SELECT, TEXTAREA, and BUTTON.

- Bubbles: No
- Cancelable: No
- Context Info: None

resize

The resize event occurs when a document view is resized.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

scroll

The scroll event occurs when a document view is scrolled.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

7. Document Object Model Traversal

Editors
Mike Champion, Software AG
Joe Kesselman, IBM
Jonathan Robie, Software AG

7.1. Overview

This chapter describes the optional DOM Level 2 *Traversal* feature. Its `TreeWalker` [p.258] , `NodeIterator` [p.253] , and `NodeFilter` [p.255] interfaces provide easy-to-use, robust, selective traversal of a document's contents. A DOM application can use the `hasFeature` method of the `DOMImplementation` [p.26] interface to determine whether this feature is supported or not. The feature string for all the interfaces listed in this section is "Traversal".

`NodeIterators` [p.253] and `TreeWalkers` [p.258] are two different ways of representing the nodes of a document subtree and a position within the nodes they present. A `NodeIterator` [p.253] presents a flattened view of the subtree as an ordered sequence of nodes, presented in document order. Because this view is presented without respect to hierarchy, iterators have methods to move forward and backward, but not to move up and down. Conversely, a `TreeWalker` [p.258] maintains the hierarchical relationships of the subtree, allowing navigation of this hierarchy. In general, `TreeWalkers` are better for tasks in which the structure of the document around selected nodes will be manipulated, while `NodeIterators` are better for tasks that focus on the content of each selected node.

`NodeIterators` [p.253] and `TreeWalkers` [p.258] each present a view of a document subtree that may not contain all nodes found in the subtree. In this specification, we refer to this as the *logical view* to distinguish it from the *physical view*, which corresponds to the document subtree per se. When an iterator or `TreeWalker` [p.258] is created, it may be associated with a `NodeFilter` [p.255] , which examines each node and determines whether it should appear in the logical view. In addition, flags may be used to specify which node types should occur in the logical view.

`NodeIterators` [p.253] and `TreeWalkers` [p.258] are dynamic - the logical view changes to reflect changes made to the underlying document. However, they differ in how they respond to those changes. `NodeIterators` [p.253] , which present the nodes sequentially, attempt to maintain their location relative to a position in that sequence when the sequence's contents change. `TreeWalkers` [p.258] , which present the nodes as a filtered tree, maintain their location relative to their current node and remain attached to that node if it is moved to a new context. We will discuss these behaviors in greater detail below.

7.1.1. `NodeIterators`

A `NodeIterator` [p.253] allows the members of a list of nodes to be returned sequentially. In the current DOM interfaces, this list will always consist of the nodes of a subtree, presented in document order. When an iterator is first created, calling its `nextNode()` method returns the first node in the logical view of the subtree; in most cases, this is the root of the subtree. Each successive call advances the `NodeIterator` through the list, returning the next node available in the logical view. When no more nodes are visible, `nextNode()` returns `null`.

`NodeIterators` [p.253] are created using the `createNodeIterator` method found in the `DocumentTraversal` [p.261] interface. When a `NodeIterator` [p.253] is created, flags can be used to determine which node types will be "visible" and which nodes will be "invisible" while traversing the tree; these flags can be combined using the `OR` operator. Nodes that are "invisible" are skipped over by the iterator as though they did not exist.

The following code creates an iterator, then calls a function to print the name of each element:

```
     NodeIterator iter=
  ((DocumentTraversal)document).createNodeIterator(
       root, NodeFilter.SHOW_ELEMENT, null);

     while (Node n = iter.nextNode())
         printMe(n);
```

7.1.1.1. Moving Forward and Backward

`NodeIterators` [p.253] present nodes as an ordered list, and move forward and backward within this list. The iterator’s position is always either between two nodes, before the first node, or after the last node. When an iterator is first created, the position is set before the first item. The following diagram shows the list view that an iterator might provide for a particular subtree, with the position indicated by an asterisk ’*’ :

```
 * A B C D E F G H I
```

Each call to `nextNode()` returns the next node and advances the position. For instance, if we start with the above position, the first call to `nextNode()` returns "A" and advances the iterator:

```
 [A] * B C D E F G H I
```

The position of a `NodeIterator` [p.253] can best be described with respect to the last node returned, which we will call the *reference node*. When an iterator is created, the first node is the reference node, and the iterator is positioned before the reference node. In these diagrams, we use square brackets to indicate the reference node.

A call to `previousNode()` returns the previous node and moves the position backward. For instance, if we start with the `NodeIterator` [p.253] between "A" and "B", it would return "A" and move to the position shown below:

```
 * [A] B C D E F G H I
```

If `nextNode()` is called at the end of a list, or `previousNode()` is called at the beginning of a list, it returns `null` and does not change the position of the iterator. When a `NodeIterator` [p.253] is first created, the reference node is the first node:

```
 * [A] B C D E F G H I
```

7.1.1.2. Robustness

A `NodeIterator` [p.253] may be active while the data structure it navigates is being edited, so an iterator must behave gracefully in the face of change. Additions and removals in the underlying data structure do not invalidate a `NodeIterator`; in fact, a `NodeIterator` is never invalidated unless its `detach()` method is invoked. To make this possible, the iterator uses the reference node to maintain its position. The state of an iterator also depends on whether the iterator is positioned before or after the reference node.

If changes to the iterated list do not remove the reference node, they do not affect the state of the `NodeIterator` [p.253] . For instance, the iterator's state is not affected by inserting new nodes in the vicinity of the iterator or removing nodes other than the reference node. Suppose we start from the following position:

```
A B C [D] * E F G H I
```

Now let's remove "E". The resulting state is:

```
A B C [D] * F G H I
```

If a new node is inserted, the `NodeIterator` [p.253] stays close to the reference node, so if a node is inserted between "D" and "F", it will occur between the iterator and "F":

```
A B C [D] * X F G H I
```

Moving a node is equivalent to a removal followed by an insertion. If we move "I" to the position before "X" the result is:

```
A B C [D] * I X F G H
```

If the reference node is removed from the list being iterated over, a different node is selected as the reference node. If the reference node's position is before that of the `NodeIterator` [p.253] , which is usually the case after `nextNode()` has been called, the nearest node before the iterator is chosen as the new reference node. Suppose we remove the "D" node, starting from the following state:

```
A B C [D] * F G H I
```

The "C" node becomes the new reference node, since it is the nearest node to the `NodeIterator` [p.253] that is before the iterator:

```
A B [C] * F G H I
```

If the reference node is after the `NodeIterator` [p.253] , which is usually the case after `previousNode()` has been called, the nearest node after the iterator is chosen as the new reference node. Suppose we remove "E", starting from the following state:

```
A B C D * [E] F G H I
```

The "F" node becomes the new reference node, since it is the nearest node to the `NodeIterator` [p.253] that is after the iterator:

```
A B C D * [F] G H I
```

As noted above, moving a node is equivalent to a removal followed by an insertion. Suppose we wish to move the "D" node to the end of the list, starting from the following state:

```
A B C [D] * F G H I C
```

The resulting state is as follows:

```
A B [C] * F G H I D
```

One special case arises when the reference node is the last node in the list and the reference node is removed. Suppose we remove node "C", starting from the following state:

```
A B * [C]
```

According to the rules we have given, the new reference node should be the nearest node after the `NodeIterator` [p.253] , but there are no further nodes after "C". The same situation can arise when `previousNode()` has just returned the first node in the list, which is then removed. Hence: If there is no node in the original direction of the reference node, the nearest node in the opposite direction is selected as the reference node:

```
A [B] *
```

If the `NodeIterator` [p.253] is positioned within a block of nodes that is removed, the above rules clearly indicate what is to be done. For instance, suppose "C" is the parent node of "D", "E", and "F", and we remove "C", starting with the following state:

```
A B C [D] * E F G H I D
```

The resulting state is as follows:

```
A [B] * G H I D
```

Finally, note that removing a `NodeIterator` [p.253] ’s `root` node from its parent does not alter the list being iterated over, and thus does not change the iterator’s state.

7.1.1.3. Visibility of Nodes

The underlying data structure that is being iterated may contain nodes that are not part of the logical view, and therefore will not be returned by the `NodeIterator` [p.253] . If nodes that are to be excluded because of the value of the `whatToShow` flag, `nextNode()` returns the next visible node, skipping over the excluded "invisible" nodes. If a `NodeFilter` [p.255] is present, it is applied before returning a node; if the filter does not accept the node, the process is repeated until a node is accepted by the filter and is returned. If no visible nodes are encountered, a `null` is returned and the iterator is positioned at the end of the list. In this case, the reference node is the last node in the list, whether or not it is visible. The same approach is taken, in the opposite direction, for `previousNode()`.

In the following examples, we will use lowercase letters to represent nodes that are in the data structure, but which are not in the logical view. For instance, consider the following list:

```
A [B] * c d E F G
```

A call to `nextNode()` returns E and advances to the following position:

```
A B c d [E] * F G
```

Nodes that are not visible may nevertheless be used as reference nodes if a reference node is removed. Suppose node "E" is removed, started from the state given above. The resulting state is:

```
A B c [d] * F G
```

Suppose a new node "X", which is visible, is inserted before "d". The resulting state is:

```
A B c X [d] * F G
```

Note that a call to `previousNode()` now returns node X. It is important not to skip over invisible nodes when the reference node is removed, because there are cases, like the one just given above, where the wrong results will be returned. When "E" was removed, if the new reference node had been "B" rather than "d", calling `previousNode()` would not return "X".

7.1.2. `NodeFilters`

`NodeFilters` [p.255] allow the user to create objects that "filter out" nodes. Each filter contains a user-written function that looks at a node and determines whether or not it should be presented as part of the traversal's logical view of the document. To use a `NodeFilter` [p.255] , you create a `NodeIterator` [p.253] or a `TreeWalker` [p.258] that uses the filter. The traversal engine applies the filter to each node, and if the filter does not accept the node, traversal skips over the node as though it were not present in the document. `NodeFilters` need not know how to navigate the structure that contains the nodes on which they operate.

Filters will be consulted when a traversal operation is performed, or when a `NodeIterator` [p.253] 's reference node is removed from the subtree being iterated over and it must select a new one. However, the exact timing of these filter calls may vary from one DOM implementation to another. For that reason, `NodeFilters` [p.255] should not attempt to maintain state based on the history of past invocations; the resulting behavior may not be portable.

Similarly, `TreeWalkers` [p.258] and `NodeIterators` [p.253] should behave as if they have no memory of past filter results, and no anticipation of future results. If the conditions a `NodeFilter` [p.255] is examining have changed (e.g., an attribute which it tests has been added or removed) since the last time the traversal logic examined this node, this change in visibility will be discovered only when the next traversal operation is performed. For example: if the filtering for the current node changes from `FILTER_SHOW` to `FILTER_SKIP`, a `TreeWalker` [p.258] will be able to navigate off that node in any direction, but not back to it unless the filtering conditions change again. `NodeFilters` which change during a traversal can be written, but their behavior may be confusing and they should be avoided when possible.

7.1.2.1. Using `NodeFilters`

A `NodeFilter` [p.255] contains one method named `acceptNode()`, which allows a `NodeIterator` [p.253] or `TreeWalker` [p.258] to pass a `Node` [p.38] to a filter and ask whether it should be present in the logical view. The `acceptNode()` function returns one of three values to state how the `Node` should be treated. If `acceptNode()` returns `FILTER_ACCEPT`, the `Node` will be present in the logical view; if it returns `FILTER_SKIP`, the `Node` will not be present in the logical view, but the children of the `Node` may; if it returns `FILTER_REJECT`, neither the `Node` nor its descendants will be present in the logical view. Since iterators present nodes as an ordered list, without hierarchy, `FILTER_REJECT` and `FILTER_SKIP` are synonyms for `NodeIterators`, skipping only the single current node.

Consider a filter that accepts the named anchors in an HTML document. In HTML, an HREF can refer to any A element that has a NAME attribute. Here is a `NodeFilter` [p.255] in Java that looks at a node and determines whether it is a named anchor:

```
class NamedAnchorFilter implements NodeFilter
{
 short acceptNode(Node n) {
  if (n.getNodeType()==Node.ELEMENT_NODE) {
   Element e = (Element)n;
   if (! e.getNodeName().equals("A"))
    return FILTER_SKIP;
  if (e.getAttributeNode("NAME") != null)
    return FILTER_ACCEPT;
   }
        return FILTER_SKIP;
  }
}
```

If the above `NodeFilter` [p.255] were to be used only with `NodeIterators` [p.253] , it could have used `FILTER_REJECT` wherever `FILTER_SKIP` is used, and the behavior would not change. For `TreeWalker` [p.258] , though, `FILTER_REJECT` would reject the children of any element that is not a named anchor, and since named anchors are always contained within other elements, this would have meant that no named anchors would be found. `FILTER_SKIP` rejects the given node, but continues to examine the children; therefore, the above filter will work with either a `NodeIterator` [p.253] or a `TreeWalker`.

To use this filter, the user would create an instance of the `NodeFilter` [p.255] and create a `NodeIterator` [p.253] using it:

```
NamedAnchorFilter myFilter = new NamedAnchorFilter();
NodeIterator iter=
     ((DocumentTraversal)document).createNodeIterator(
          node, NodeFilter.SHOW_ELEMENT, myFilter);
```

Note that the use of the `SHOW_ELEMENT` flag is not strictly necessary in this example, since our sample `NodeFilter` [p.255] tests the `nodeType`. However, some implementations of the Traversal interfaces may be able to improve `whatToShow` performance by taking advantage of knowledge of the document's structure, which makes the use of `SHOW_ELEMENT` worthwhile. Conversely, while we could remove the `nodeType` test from our filter, that would make it dependent upon `whatToShow` to distinguish between `Elements` [p.57] , `Attr` [p.56] ’s, and `ProcessingInstructions` [p.71] .

7.1.2.2. `NodeFilters` and Exceptions

When writing a `NodeFilter` [p.255] , users should avoid writing code that can throw an exception. However, because a DOM implementation can not prevent exceptions from being thrown, it is important that the behavior of filters that throw an exception be well-defined. A `TreeWalker` [p.258] or `NodeIterator` [p.253] does not catch or alter an exception thrown by a filter, but lets it propagate up to the user’s code. The following functions may invoke a `NodeFilter`, and may therefore propagate an exception if one is thrown by a filter:

1. `NodeIterator` [p.253] `.nextNode()`
2. `NodeIterator` [p.253] `.previousNode()`
3. `TreeWalker` [p.258] `.firstChild()`
4. `TreeWalker` [p.258] `.lastChild()`
5. `TreeWalker` [p.258] `.nextSibling()`
6. `TreeWalker` [p.258] `.previousSibling()`
7. `TreeWalker` [p.258] `.nextNode()`
8. `TreeWalker` [p.258] `.previousNode()`
9. `TreeWalker` [p.258] `.parentNode()`

7.1.2.3. `NodeFilters` and Document Mutation

Well-designed `NodeFilters` [p.255] should not have to modify the underlying structure of the document. But a DOM implementation can not prevent a user from writing filter code that does alter the document structure. Traversal does not provide any special processing to handle this case. For instance, if a `NodeFilter` [p.255] removes a node from a document, it can still accept the node, which means that the node may be returned by the `NodeIterator` [p.253] or `TreeWalker` [p.258] even though it is no longer in the subtree being traversed. In general, this may lead to inconsistent, confusing results, so we encourage users to write `NodeFilters` that make no changes to document structures. Instead, do your editing in the loop controlled by the traversal object.

7.1.2.4. `NodeFilters` and `whatToShow` flags

`NodeIterator` [p.253] and `TreeWalker` [p.258] apply their `whatToShow` flags before applying filters. If a node is skipped by the active `whatToShow` flags, a `NodeFilter` [p.255] will not be called to evaluate that node. Please note that this behavior is similar to that of `FILTER_SKIP`; children of that node will be considered, and filters may be called to evaluate them. Also note that it will in fact be a "skip" even if the `NodeFilter` would have preferred to reject the entire subtree; if this would cause a problem in your application, consider setting `whatToShow` to `SHOW_ALL` and performing the `nodeType` test inside your filter.

7.1.3. `TreeWalker`

The `TreeWalker` [p.258] interface provides many of the same benefits as the `NodeIterator` [p.253] interface. The main difference between these two interfaces is that the `TreeWalker` presents a tree-oriented view of the nodes in a subtree, rather than the iterator's list-oriented view. In other words, an iterator allows you to move forward or back, but a `TreeWalker` allows you to also move to the parent of a node, to one of its children, or to a sibling.

Using a `TreeWalker` [p.258] is quite similar to navigation using the Node directly, and the navigation methods for the two interfaces are analogous. For instance, here is a function that recursively walks over a tree of nodes in document order, taking separate actions when first entering a node and after processing any children:

```
processMe(Node n) {
   nodeStartActions(n);
   for (Node child=n.firstChild();
        child != null;
        child=child.nextSibling()) {
      processMe(child);
   }
   nodeEndActions(n);
}
```

Doing the same thing using a `TreeWalker` [p.258] is quite similar. There is one difference: since navigation on the `TreeWalker` changes the current position, the position at the end of the function has changed. A read/write attribute named `currentNode` allows the current node for a `TreeWalker` to be both queried and set. We will use this to ensure that the position of the `TreeWalker` is restored when this function is completed:

```
processMe(TreeWalker tw) {
   Node n = tw.getCurrentNode();
   nodeStartActions(tw);
   for (Node child=tw.firstChild();
        child!=null;
        child=tw.nextSibling()) {
      processMe(tw);
   }

   tw.setCurrentNode(n);
   nodeEndActions(tw);
}
```

The advantage of using a `TreeWalker` [p.258] instead of direct `Node` [p.38] navigation is that the `TreeWalker` allows the user to choose an appropriate view of the tree. Flags may be used to show or hide `Comments` [p.66] or `ProcessingInstructions` [p.71] ; entities may be expanded or shown as `EntityReference` [p.70] nodes. In addition, `NodeFilters` [p.255] may be used to present a custom view of the tree. Suppose a program needs a view of a document that shows which tables occur in each chapter, listed by chapter. In this view, only the chapter elements and the tables that they contain are seen. The first step is to write an appropriate filter:

```
class TablesInChapters implements NodeFilter {

   short acceptNode(Node n) {
      if (n.getNodeType()==Node.ELEMENT_NODE) {

          if (n.getNodeName().equals("CHAPTER"))
             return FILTER_ACCEPT;

          if (n.getNodeName().equals("TABLE"))
             return FILTER_ACCEPT;

          if (n.getNodeName().equals("SECT1")
              || n.getNodeName().equals("SECT2")
              || n.getNodeName().equals("SECT3")
              || n.getNodeName().equals("SECT4")
              || n.getNodeName().equals("SECT5")
              || n.getNodeName().equals("SECT6")
              || n.getNodeName().equals("SECT7"))
             return FILTER_SKIP;

      }

      return FILTER_REJECT;
        }
}
```

This filter assumes that TABLE elements are contained directly in CHAPTER or SECTn elements. If another kind of element is encountered, it and its children are rejected. If a SECTn element is encountered, it is skipped, but its children are explored to see if they contain any TABLE elements.

Now the program can create an instance of this `NodeFilter` [p.255] , create a `TreeWalker` [p.258] that uses it, and pass this `TreeWalker` to our ProcessMe() function:

```
TablesInChapters tablesInChapters  = new TablesInChapters();
TreeWalker tw  =
     ((DocumentTraversal)document).createTreeWalker(
          root, NodeFilter.SHOW_ELEMENT, tablesInChapters);
processMe(tw);
```

(Again, we’ve chosen to both test the `nodeType` in the filter’s logic and use `SHOW_ELEMENT`, for the reasons discussed in the earlier `NodeIterator` [p.253] example.)

Without making any changes to the above `ProcessMe()` function, it now processes only the CHAPTER and TABLE elements. The programmer can write other filters or set other flags to choose different sets of nodes; if functions use `TreeWalker` [p.258] to navigate, they will support any view of the document defined with a `TreeWalker`.

Note that the structure of a `TreeWalker` [p.258] ’s filtered view of a document may differ significantly from that of the document itself. For example, a `TreeWalker` with only `SHOW_TEXT` specified in its `whatToShow` parameter would present all the `Text` [p.66] nodes as if they were siblings of each other yet had no parent.

7.1.3.1. Robustness

As with `NodeIterators` [p.253] , a `TreeWalker` [p.258] may be active while the data structure it navigates is being edited, and must behave gracefully in the face of change. Additions and removals in the underlying data structure do not invalidate a `TreeWalker`; in fact, a `TreeWalker` is never invalidated.

But a `TreeWalker` [p.258] 's response to these changes is quite different from that of a `NodeIterator` [p.253] . While `NodeIterators` respond to editing by maintaining their position within the list that they are iterating over, `TreeWalkers` will instead remain attached to their `currentNode`. All the `TreeWalker`'s navigation methods operate in terms of the context of the `currentNode` at the time they are invoked, no matter what has happened to, or around, that node since the last time the `TreeWalker` was accessed. This remains true even if the `currentNode` is moved out of its original subtree.

As an example, consider the following document fragment:

```
...
<subtree>
        <twRoot>
                <currentNode/>
                <anotherNode/>
        </twRoot>
</subtree>
...
```

Let's say we have created a `TreeWalker` [p.258] whose `root` node is the <twRoot/> element and whose `currentNode` is the <currentNode/> element. For this illustration, we will assume that all the nodes shown above are accepted by the `TreeWalker`'s `whatToShow` and filter settings.

If we use `removeChild()` to remove the <currentNode/> element from its parent, that element remains the `TreeWalker` [p.258] 's `currentNode`, even though it is no longer within the `root` node's subtree. We can still use the `TreeWalker` to navigate through any children that the orphaned `currentNode` may have, but are no longer able to navigate outward from the `currentNode` since there is no parent available.

If we use `insertBefore()` or `appendChild()` to give the <currentNode/> a new parent, then `TreeWalker` [p.258] navigation will operate from the `currentNode`'s new location. For example, if we inserted the <currentNode/> immediately after the <anotherNode/> element, the `TreeWalker`'s `previousSibling()` operation would move it back to the <anotherNode/>, and calling `parentNode()` would move it up to the <twRoot/>.

If we instead insert the `currentNode` into the <subtree/> element, like so:

```
...
<subtree>
        <currentNode/>
        <twRoot>
                <anotherNode/>
        </twRoot>
</subtree>
...
```

we have moved the `currentNode` out from under the `TreeWalker` [p.258] 's `root` node. This does not invalidate the `TreeWalker`; it may still be used to navigate relative to the `currentNode`. Calling its `parentNode()` operation, for example, would move it to the <subtree/> element, even though that too is outside the original `root` node. However, if the `TreeWalker`'s navigation should take it back into the original `root` node's subtree -- for example, if rather than calling `parentNode()` we called `nextNode()`, moving the `TreeWalker` to the <twRoot/> element -- the `root` node will "recapture" the `TreeWalker`, and prevent it from traversing back out.

This becomes a bit more complicated when filters are in use. Relocation of the `currentNode` -- or explicit selection of a new `currentNode`, or changes in the conditions that the `NodeFilter` [p.255] is basing its decisions on -- can result in a `TreeWalker` [p.258] having a `currentNode` which would not otherwise be visible in the filtered (logical) view of the document. This node can be thought of as a "transient member" of that view. When you ask the `TreeWalker` to navigate off this node the result will be just as if it had been visible, but you may be unable to navigate back to it unless conditions change to make it visible again.

In particular: If the `currentNode` becomes part of a subtree that would otherwise have been Rejected by the filter, that entire subtree may be added as transient members of the logical view. You will be able to navigate within that subtree (subject to all the usual filtering) until you move upward past the Rejected ancestor. The behavior is as if the Rejected node had only been Skipped (since we somehow wound up inside its subtree) until we leave it; thereafter, standard filtering applies.

7.2. Formal Interface Definition

Interface *NodeIterator* (introduced in **DOM Level 2**)

`Iterators` are used to step through a set of nodes, e.g. the set of nodes in a `NodeList` [p.47] , the document subtree governed by a particular `Node` [p.38] , the results of a query, or any other set of nodes. The set of nodes to be iterated is determined by the implementation of the `NodeIterator`. DOM Level 2 specifies a single `NodeIterator` implementation for document-order traversal of a document subtree. Instances of these iterators are created by calling `DocumentTraversal` [p.261] `.createNodeIterator()`.

IDL Definition

```
// Introduced in DOM Level 2:
interface NodeIterator {
  readonly attribute Node             root;
  readonly attribute unsigned long    whatToShow;
  readonly attribute NodeFilter       filter;
  readonly attribute boolean          expandEntityReferences;
  Node               nextNode()
                                        raises(DOMException);
  Node               previousNode()
                                        raises(DOMException);
  void               detach();
};
```

Attributes

`expandEntityReferences` of type `boolean`, readonly

The value of this flag determines whether the children of entity reference nodes are visible to the iterator. If false, they and their descendents will be rejected. Note that this rejection takes precedence over `whatToShow` and the filter. Also note that this is currently the only situation where `NodeIterators` may reject a complete subtree rather than skipping individual nodes.

To produce a view of the document that has entity references expanded and does not expose the entity reference node itself, use the `whatToShow` flags to hide the entity reference node and set `expandEntityReferences` to true when creating the iterator. To produce a view of the document that has entity reference nodes but no entity expansion, use the `whatToShow` flags to show the entity reference node and set `expandEntityReferences` to false.

`filter` of type `NodeFilter` [p.255] , readonly

The `NodeFilter` [p.255] used to screen nodes.

`root` of type `Node` [p.38] , readonly

The root node of the `NodeIterator`, as specified when it was created.

`whatToShow` of type `unsigned long`, readonly

This attribute determines which node types are presented via the iterator. The available set of constants is defined in the `NodeFilter` [p.255] interface. Nodes not accepted by `whatToShow` will be skipped, but their children may still be considered. Note that this skip takes precedence over the filter, if any.

Methods

`detach`

Detaches the `NodeIterator` from the set which it iterated over, releasing any computational resources and placing the iterator in the INVALID state. After `detach` has been invoked, calls to `nextNode` or `previousNode` will raise the exception INVALID_STATE_ERR.

No Parameters

No Return Value

No Exceptions

`nextNode`

Returns the next node in the set and advances the position of the iterator in the set. After a `NodeIterator` is created, the first call to `nextNode()` returns the first node in the set.

Return Value

`Node` [p.38]	The next `Node` in the set being iterated over, or `null` if there are no more members in that set.

Exceptions

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if this method is called after the `detach` method was invoked.

No Parameters

`previousNode`

Returns the previous node in the set and moves the position of the `NodeIterator` backwards in the set.

Return Value

`Node` [p.38]	The previous `Node` in the set being iterated over, or `null` if there are no more members in that set.

Exceptions

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if this method is called after the `detach` method was invoked.

No Parameters

Interface *NodeFilter* (introduced in **DOM Level 2**)

Filters are objects that know how to "filter out" nodes. If a `NodeIterator` [p.253] or `TreeWalker` [p.258] is given a `NodeFilter`, it applies the filter before it returns the next node. If the filter says to accept the node, the traversal logic returns it; otherwise, traversal looks for the next node and pretends that the node that was rejected was not there.

The DOM does not provide any filters. `NodeFilter` is just an interface that users can implement to provide their own filters.

`NodeFilters` do not need to know how to traverse from node to node, nor do they need to know anything about the data structure that is being traversed. This makes it very easy to write filters, since the only thing they have to know how to do is evaluate a single node. One filter may be used with a number of different kinds of traversals, encouraging code reuse.

IDL Definition

```
// Introduced in DOM Level 2:
interface NodeFilter {
  // Constants returned by acceptNode
  const short              FILTER_ACCEPT                  = 1;
  const short              FILTER_REJECT                  = 2;
  const short              FILTER_SKIP                    = 3;

  // Constants for whatToShow
  const unsigned long      SHOW_ALL                       = 0xFFFFFFFF;
  const unsigned long      SHOW_ELEMENT                   = 0x00000001;
  const unsigned long      SHOW_ATTRIBUTE                 = 0x00000002;
  const unsigned long      SHOW_TEXT                      = 0x00000004;
  const unsigned long       SHOW_CDATA_SECTION             = 0x00000008;
  const unsigned long       SHOW_ENTITY_REFERENCE          = 0x00000010;
  const unsigned long       SHOW_ENTITY                    = 0x00000020;
  const unsigned long       SHOW_PROCESSING_INSTRUCTION    = 0x00000040;
  const unsigned long       SHOW_COMMENT                   = 0x00000080;
  const unsigned long       SHOW_DOCUMENT                  = 0x00000100;
  const unsigned long       SHOW_DOCUMENT_TYPE             = 0x00000200;
  const unsigned long       SHOW_DOCUMENT_FRAGMENT         = 0x00000400;
  const unsigned long       SHOW_NOTATION                  = 0x00000800;

  short              acceptNode(in Node n);
};
```

Definition group *Constants returned by acceptNode*

The following constants are returned by the acceptNode() method:

Defined Constants

`FILTER_ACCEPT`
: Accept the node. Navigation methods defined for `NodeIterator` [p.253] or `TreeWalker` [p.258] will return this node.

`FILTER_REJECT`
: Reject the node. Navigation methods defined for `NodeIterator` [p.253] or `TreeWalker` [p.258] will not return this node. For `TreeWalker`, the children of this node will also be rejected. `NodeIterators` treat this as a synonym for `FILTER_SKIP`.

`FILTER_SKIP`
: Skip this single node. Navigation methods defined for `NodeIterator` [p.253] or `TreeWalker` [p.258] will not return this node. For both `NodeIterator` and `TreeWalker`, the children of this node will still be considered.

Definition group *Constants for whatToShow*

These are the available values for the `whatToShow` parameter used in `TreeWalkers` [p.258] and `NodeIterators` [p.253] . They are the same as the set of possible types for `Node` [p.38] , and their values are derived by using a bit position corresponding to the value of `nodeType` for the equivalent node type. If a bit in `whatToShow` is set false, that will be taken as a request to skip over this type of node; the behavior in that case is similar to that of `FILTER_SKIP`.

Note that if node types greater than 32 are ever introduced, they may not be individually testable via `whatToShow`. If that need should arise, it can be handled by selecting `SHOW_ALL` together with an appropriate `NodeFilter`.

Defined Constants

`SHOW_ALL`
: Show all `Nodes` [p.38] .

`SHOW_ATTRIBUTE`
: Show `Attr` [p.56] nodes. This is meaningful only when creating an iterator or tree-walker with an attribute node as its `root`; in this case, it means that the attribute

node will appear in the first position of the iteration or traversal. Since attributes are never children of other nodes, they do not appear when traversing over the document tree.

`SHOW_CDATA_SECTION`
: Show `CDATASection` [p.67] nodes.

`SHOW_COMMENT`
: Show `Comment` [p.66] nodes.

`SHOW_DOCUMENT`
: Show `Document` [p.29] nodes.

`SHOW_DOCUMENT_FRAGMENT`
: Show `DocumentFragment` [p.28] nodes.

`SHOW_DOCUMENT_TYPE`
: Show `DocumentType` [p.68] nodes.

`SHOW_ELEMENT`
: Show `Element` [p.57] nodes.

`SHOW_ENTITY`
: Show `Entity` [p.69] nodes. This is meaningful only when creating an iterator or tree-walker with an `Entity` node as its `root`; in this case, it means that the `Entity` node will appear in the first position of the traversal. Since entities are not part of the document tree, they do not appear when traversing over the document tree.

`SHOW_ENTITY_REFERENCE`
: Show `EntityReference` [p.70] nodes.

`SHOW_NOTATION`
: Show `Notation` [p.69] nodes. This is meaningful only when creating an iterator or tree-walker with a `Notation` node as its `root`; in this case, it means that the `Notation` node will appear in the first position of the traversal. Since notations are not part of the document tree, they do not appear when traversing over the document tree.

`SHOW_PROCESSING_INSTRUCTION`
: Show `ProcessingInstruction` [p.71] nodes.

`SHOW_TEXT`
: Show `Text` [p.66] nodes.

Methods

`acceptNode`
: Test whether a specified node is visible in the logical view of a `TreeWalker` [p.258] or `NodeIterator` [p.253] . This function will be called by the implementation of

`TreeWalker` and `NodeIterator`; it is not normally called directly from user code. (Though you could do so if you wanted to use the same filter to guide your own application logic.)

Parameters

`n` of type `Node` [p.38]
The node to check to see if it passes the filter or not.

Return Value

`short`	a constant to determine whether the node is accepted, rejected, or skipped, as defined above [p.256] .

No Exceptions

Interface *TreeWalker* (introduced in **DOM Level 2**)

`TreeWalker` objects are used to navigate a document tree or subtree using the view of the document defined by their `whatToShow` flags and filter (if any). Any function which performs navigation using a `TreeWalker` will automatically support any view defined by a `TreeWalker`.

Omitting nodes from the logical view of a subtree can result in a structure that is substantially different from the same subtree in the complete, unfiltered document. Nodes that are siblings in the `TreeWalker` view may be children of different, widely separated nodes in the original view. For instance, consider a `NodeFilter` [p.255] that skips all nodes except for Text nodes and the root node of a document. In the logical view that results, all text nodes will be siblings and appear as direct children of the root node, no matter how deeply nested the structure of the original document.

IDL Definition

```
// Introduced in DOM Level 2:
interface TreeWalker {
  readonly attribute Node             root;
  readonly attribute unsigned long    whatToShow;
  readonly attribute NodeFilter       filter;
  readonly attribute boolean          expandEntityReferences;
           attribute Node             currentNode;
                                        // raises(DOMException) on setting

  Node               parentNode();
  Node               firstChild();
  Node               lastChild();
  Node               previousSibling();
  Node               nextSibling();
  Node               previousNode();
  Node               nextNode();
};
```

Attributes

`currentNode` of type `Node` [p.38]
The node at which the `TreeWalker` is currently positioned.
Alterations to the DOM tree may cause the current node to no longer be accepted by the

`TreeWalker`'s associated filter. `currentNode` may also be explicitly set to any node, whether or not it is within the subtree specified by the `root` node or would be accepted by the filter and `whatToShow` flags. Further traversal occurs relative to `currentNode` even if it is not part of the current view, by applying the filters in the requested direction; if no traversal is possible, `currentNode` is not changed.

Exceptions on setting

`DOMException` [p.24]	NOT_SUPPORTED_ERR: Raised if an attempt is made to set `currentNode` to `null`.

`expandEntityReferences` of type `boolean`, readonly

The value of this flag determines whether the children of entity reference nodes are visible to the `TreeWalker`. If false, they and their descendents will be rejected. Note that this rejection takes precedence over `whatToShow` and the filter, if any.

To produce a view of the document that has entity references expanded and does not expose the entity reference node itself, use the `whatToShow` flags to hide the entity reference node and set `expandEntityReferences` to true when creating the `TreeWalker`. To produce a view of the document that has entity reference nodes but no entity expansion, use the `whatToShow` flags to show the entity reference node and set `expandEntityReferences` to false.

`filter` of type `NodeFilter` [p.255] , readonly

The filter used to screen nodes.

`root` of type `Node` [p.38] , readonly

The `root` node of the `TreeWalker`, as specified when it was created.

`whatToShow` of type `unsigned long`, readonly

This attribute determines which node types are presented via the `TreeWalker`. The available set of constants is defined in the `NodeFilter` [p.255] interface. Nodes not accepted by `whatToShow` will be skipped, but their children may still be considered. Note that this skip takes precedence over the filter, if any.

Methods

`firstChild`

Moves the `TreeWalker` to the first visible child of the current node, and returns the new node. If the current node has no visible children, returns `null`, and retains the current node.

Return Value

`Node` [p.38]	The new node, or `null` if the current node has no visible children in the `TreeWalker`'s logical view.

No Parameters

No Exceptions

`lastChild`

Moves the `TreeWalker` to the last visible child of the current node, and returns the new node. If the current node has no visible children, returns `null`, and retains the current node.

Return Value

`Node` [p.38]	The new node, or `null` if the current node has no children in the `TreeWalker`'s logical view.

No Parameters

No Exceptions

`nextNode`

Moves the `TreeWalker` to the next visible node in document order relative to the current node, and returns the new node. If the current node has no next node, or if the search for nextNode attempts to step upward from the `TreeWalker`'s `root` node, returns `null`, and retains the current node.

Return Value

`Node` [p.38]	The new node, or `null` if the current node has no next node in the `TreeWalker`'s logical view.

No Parameters

No Exceptions

`nextSibling`

Moves the `TreeWalker` to the next sibling of the current node, and returns the new node. If the current node has no visible next sibling, returns `null`, and retains the current node.

Return Value

`Node` [p.38]	The new node, or `null` if the current node has no next sibling in the `TreeWalker`'s logical view.

No Parameters

No Exceptions

`parentNode`

Moves to and returns the closest visible ancestor node of the current node. If the search for `parentNode` attempts to step upward from the `TreeWalker`'s `root` node, or if it fails to find a visible ancestor node, this method retains the current position and returns `null`.

Return Value

`Node` [p.38]	The new parent node, or `null` if the current node has no parent in the `TreeWalker`'s logical view.

No Parameters
No Exceptions

`previousNode`

Moves the `TreeWalker` to the previous visible node in document order relative to the current node, and returns the new node. If the current node has no previous node, or if the search for `previousNode` attempts to step upward from the `TreeWalker`'s `root` node, returns `null`, and retains the current node.

Return Value

`Node` [p.38]	The new node, or `null` if the current node has no previous node in the `TreeWalker`'s logical view.

No Parameters
No Exceptions

`previousSibling`

Moves the `TreeWalker` to the previous sibling of the current node, and returns the new node. If the current node has no visible previous sibling, returns `null`, and retains the current node.

Return Value

`Node` [p.38]	The new node, or `null` if the current node has no previous sibling in the `TreeWalker`'s logical view.

No Parameters
No Exceptions

Interface *DocumentTraversal* (introduced in **DOM Level 2**)

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

IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentTraversal {
  NodeIterator       createNodeIterator(in Node root, 
                                        in unsigned long whatToShow, 
                                        in NodeFilter filter, 
                                        in boolean entityReferenceExpansion)
                                        raises(DOMException);
  TreeWalker         createTreeWalker(in Node root, 
                                      in unsigned long whatToShow, 
                                      in NodeFilter filter,
                                      in boolean entityReferenceExpansion)
                                        raises(DOMException);
};
```

Methods

`createNodeIterator`

Create a new `NodeIterator` [p.253] over the subtree rooted at the specified node.

Parameters

`root` of type `Node` [p.38]

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

`whatToShow` of type `unsigned long`

This flag specifies which node types may appear in the logical view of the tree presented by the iterator. See the description of `NodeFilter` [p.255] for the set of possible `SHOW_` values.
These flags can be combined using `OR`.

`filter` of type `NodeFilter` [p.255]

The `NodeFilter` to be used with this `TreeWalker` [p.258] , or `null` to indicate no filter.

`entityReferenceExpansion` of type `boolean`

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

Return Value

`NodeIterator` [p.253]	The newly created `NodeIterator`.

Exceptions

`DOMException` [p.24]	NOT_SUPPORTED_ERR: Raised if the specified `root` is `null`.

`createTreeWalker`

Create a new `TreeWalker` [p.258] over the subtree rooted at the specified node.

Parameters

`root` of type `Node` [p.38]

The node which will serve as the `root` for the `TreeWalker` [p.258] . The `whatToShow` flags and the `NodeFilter` [p.255] are not considered when setting this value; any node type will be accepted as the `root`. The `currentNode` of the `TreeWalker` is initialized to this node, whether or not it is visible. The `root` functions as a stopping point for traversal methods that look upward in the document structure, such as `parentNode` and nextNode. The `root` must not be `null`.

`whatToShow` of type `unsigned long`
: This flag specifies which node types may appear in the logical view of the tree presented by the tree-walker. See the description of `NodeFilter` [p.255] for the set of possible SHOW_ values.
These flags can be combined using `OR`.

`filter` of type `NodeFilter` [p.255]
: The `NodeFilter` to be used with this `TreeWalker` [p.258] , or `null` to indicate no filter.

`entityReferenceExpansion` of type `boolean`
: If this flag is false, the contents of `EntityReference` [p.70] nodes are not presented in the logical view.

Return Value

`TreeWalker` [p.258]	The newly created `TreeWalker`.

Exceptions

`DOMException` [p.24]	NOT_SUPPORTED_ERR: Raised if the specified `root` is `null`.

8. Document Object Model Range

Editors
Vidur Apparao, Netscape Communications
Peter Sharpe, SoftQuad Software Inc.

8.1. Introduction

A Range identifies a range of content in a Document, DocumentFragment or Attr. It is contiguous in the sense that it can be characterized as selecting all of the content between a pair of boundary-points.

Note: In a text editor or a word processor, a user can make a selection by pressing down the mouse at one point in a document, moving the mouse to another point, and releasing the mouse. The resulting selection is contiguous and consists of the content between the two points.

The term ’selecting’ does not mean that every Range corresponds to a selection made by a GUI user; however, such a selection can be returned to a DOM user as a Range.

Note: In bidirectional writing (Arabic, Hebrew), a range may correspond to a logical selection that is not necessarily contiguous when displayed. A visually contiguous selection, also used in some cases, may not correspond to a single logical selection, and may therefore have to be represented by more than one range.

The Range interface provides methods for accessing and manipulating the document tree at a higher level than similar methods in the Node interface. The expectation is that each of the methods provided by the Range interface for the insertion, deletion and copying of content can be directly mapped to a series of Node editing operations enabled by DOM Core. In this sense, the Range operations can be viewed as convenience methods that also enable the implementation to optimize common editing patterns.

This chapter describes the Range interface, including methods for creating and moving a Range and methods for manipulating content with Ranges. The feature string for the interfaces listed in this section is "Range".

8.2. Definitions and Notation

8.2.1. Position

This chapter refers to two different representations of a document: the text or source form that includes the document markup and the tree representation similar to the one described in the What is the Document Object Model? [p.13] section.

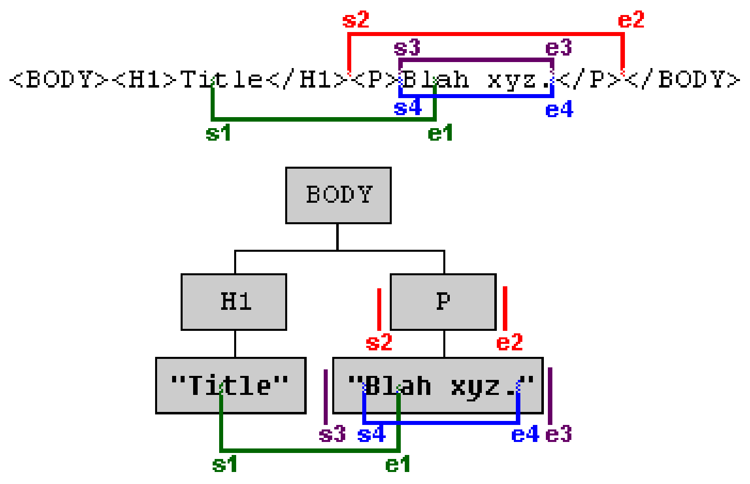
A Range consists of two *boundary-points* corresponding to the start and the end of the Range. A boundary-point’s position in a Document or DocumentFragment tree can be characterized by a node and an offset. The node is called the *container* of the boundary-point and of its position. The container and its ancestors are the *ancestor container*s of the boundary-point and of its position. The offset within the node is called the *offset* of the boundary-point and its position. If the container is an Attr, Document, DocumentFragment, Element or EntityReference node, the offset is between its child nodes. If the container is a CharacterData, Comment or ProcessingInstruction node, the offset is between the 16-bit units of the UTF-16 encoded string contained by it.

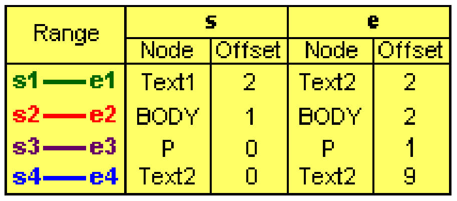
The *boundary-points* [p.265] of a Range must have a common *ancestor container* [p.265] which is either a Document, DocumentFragment or Attr node. That is, the content of a Range must be entirely within the subtree rooted by a single Document, DocumentFragment or Attr Node. This common *ancestor container* [p.265] is known as the *root container* of the Range. The tree rooted by the *root container* [p.266] is known as the Range's *context tree*.

The *container* [p.265] of an *boundary-point* [p.265] of a Range must be an Element, Comment, ProcessingInstruction, EntityReference, CDATASection, Document, DocumentFragment, Attr, or Text node. None of the *ancestor container* [p.265] s of the *boundary-point* of a Range can be a DocumentType, Entity or Notation node.

In terms of the text representation of a document, the *boundary-points* [p.265] of a Range can only be on token boundaries. That is, the *boundary-point* of the text range cannot be in the middle of a start- or end-tag of an element or within the name of an entity or character reference. A Range locates a contiguous portion of the content of the structure model.

The relationship between locations in a text representation of the document and in the Node tree interface of the DOM is illustrated in the following diagram:





Range	s		e	
	Node	Offset	Node	Offset
s1——e1	Text1	2	Text2	2
s2——e2	BODY	1	BODY	2
s3——e3	P	0	P	1
s4——e4	Text2	0	Text2	9

Range Example

In this diagram, four different Ranges are illustrated. The *boundary-points* [p.265] of each Range are labelled with *s#* (the start of the Range) and *e#* (the end of the Range), where # is the number of the Range. For Range 2, the start is in the BODY element and is immediately after the H1 element and immediately before the P element, so its position is between the H1 and P children of BODY. The *offset* [p.265] of a *boundary-point* whose *container* [p.265] is not a CharacterData node is 0 if it is before the first child, 1 if between the first and second child, and so on. So, for the start of the Range 2, the *container* is BODY and the *offset* is 1. The *offset* of a *boundary-point* whose *container* is a CharacterData node is obtained similarly but using 16-bit unit positions instead. For example, the *boundary-point* labelled s1 of the Range 1 has a Text node (the one containing "Title") as its *container* and an *offset* of 2 since it is between the second and third 16-bit unit.

Notice that the *boundary-point* [p.265] s of Ranges 3 and 4 correspond to the same location in the text representation. An important feature of the Range is that a *boundary-point* of a Range can unambiguously represent every position within the document tree.

The *container* [p.265] s and *offset* [p.265] s of the *boundary-point* [p.265] s can be obtained through the following read-only Range attributes:

```
  readonly attribute Node startContainer;
  readonly attribute long startOffset;
  readonly attribute Node endContainer;
  readonly attribute long endOffset;
```

If the *boundary-point* [p.265] s of a Range have the same *container* [p.265] s and *offset* [p.265] s, the Range is said to be a *collapsed* Range. (This is often referred to as an insertion point in a user agent.)

8.2.2. Selection and Partial Selection

A node or 16-bit unit is said to be *selected* by a Range if it is between the two *boundary-point* [p.265] s of the Range, that is, if the position immediately before the node or 16-bit unit is before the end of the Range and the position immediately after the node or 16-bit unit is after the start of the range. For example, in terms of a text representation of the document, an element would be *selected* [p.267] by a Range if its corresponding start-tag was located after the start of the Range and its end-tag was located before the end of the Range. In the examples in the above diagram, the Range 2 *selects* the P node and the Range 3 *selects* the text node containing the text "Blah xyz."

A node is said to be *partially selected* by a Range if it is an *ancestor container* [p.265] of exactly one *boundary-point* [p.265] of the Range. For example, consider Range 1 in the above diagram. The element H1 is *partially selected* [p.267] by that Range since the start of the Range is within one of its children.

8.2.3. Notation

Many of the examples in this chapter are illustrated using a text representation of a document. The *boundary-point* [p.265] s of a Range are indicated by displaying the characters (be they markup or data characters) between the two *boundary-point*s in bold, as in

```
<FOO>ABC<BAR>DEF</BAR></FOO>
```

When both *boundary-point* [p.265] s are at the same position, they are indicated with a bold caret ('^'), as in

```
<FOO>A^BC<BAR>DEF</BAR></FOO>
```

And when referring to a single *boundary-point* [p.265] , it will be shown as a bold asterisk ('*****') as in

```
<FOO>A*BC<BAR>DEF</BAR></FOO>
```

8.3. Creating a Range

A Range is created by calling the `createRange()` method on the `DocumentRange` [p.288] interface. This interface can be obtained from the object implementing the `Document` [p.29] interface using binding-specific casting methods.

```
interface DocumentRange {
  Range createRange();
}
```

The initial state of the Range returned from this method is such that both of its *boundary-point* [p.265] s are positioned at the beginning of the corresponding Document, before any content. In other words, the *container* [p.265] of each *boundary-point* is the Document node and the offset within that node is 0.

Like some objects created using methods in the Document interface (such as Nodes and DocumentFragments), Ranges created via a particular document instance can select only content associated with that Document, or with DocumentFragments and Attrs for which that Document is the `ownerDocument`. Such Ranges, then, can not be used with other Document instances.

8.4. Changing a Range's Position

A Range's position can be specified by setting the *container* [p.265] and *offset* [p.265] of each boundary-point with the `setStart` and `setEnd` methods.

```
void setStart(in Node parent, in long offset)
                      raises(RangeException);
void setEnd(in Node parent, in long offset)
              raises(RangeException);
```

If one boundary-point of a Range is set to have a *root container* [p.266] other than the current one for the Range, the Range is *collapsed* [p.267] to the new position. This enforces the restriction that both boundary-points of a Range must have the same *root container*.

The start position of a Range is guaranteed to never be after the end position. To enforce this restriction, if the start is set to be at a position after the end, the Range is *collapsed* [p.267] to that position. Similarly, if the end is set to be at a position before the start, the Range is *collapsed* to that position.

It is also possible to set a Range's position relative to nodes in the tree:

```
  void setStartBefore(in Node node);
                                raises(RangeException);
  void setStartAfter(in Node node);
                         raises(RangeException);
  void setEndBefore(in Node node);
                        raises(RangeException);
  void setEndAfter(in Node node);
                       raises(RangeException);
```

The parent of the node becomes the *container* [p.265] of the *boundary-point* [p.265] and the Range is subject to the same restrictions as given above in the description of `setStart()`and `setEnd()`.

A Range can be *collapsed* [p.267] to either boundary-point:

```
  void collapse(in boolean toStart);
```

Passing `TRUE` as the parameter `toStart` will *collapse* [p.267] the Range to its start, `FALSE` to its end.

Testing whether a Range is *collapsed* [p.267] can be done by examining the `collapsed` attribute:

```
  readonly attribute boolean collapsed;
```

The following methods can be used to make a Range select the contents of a node or the node itself.

```
  void selectNode(in Node n);
  void selectNodeContents(in Node n);
```

The following examples demonstrate the operation of the methods `selectNode` and `selectNodeContents`:

```
Before:
  ^<BAR><FOO>A<MOO>B</MOO>C</FOO></BAR>
After Range.selectNodeContents(FOO):
  <BAR><FOO>A<MOO>B</MOO>C</FOO></BAR>
(In this case, FOO is the parent of both boundary-points)
After Range.selectNode(FOO):

<BAR><FOO>A<MOO>B</MOO>C</FOO></BAR>
```

8.5. Comparing Range Boundary-Points

It is possible to compare two Ranges by comparing their boundary-points:

```
  int compareBoundaryPoints(in CompareHow how, in Range sourceRange) raises(RangeException);
```

where `CompareHow` is one of four values: `START_TO_START`, `START_TO_END`, `END_TO_END` and `END_TO_START`. The return value is -1, 0 or 1 depending on whether the corresponding boundary-point of the Range is before, equal to, or after the corresponding boundary-point of `sourceRange`. An exception is thrown if the two Ranges have different *root container* [p.266] s.

The result of comparing two boundary-points (or positions) is specified below. An informal but not always correct specification is that an boundary-point is before, equal to, or after another if it corresponds to a location in a text representation before, equal to, or after the other’s corresponding location.

Let A and B be two boundary-points or positions. Then one of the following holds: A is *before* B, A is *equal to*B, or A is *after* B. Which one holds is specified in the following by examining four cases:

In the first case the boundary-points have the same *container* [p.265] . A is *before* B if its *offset* [p.265] is less than the *offset* of B, A is *equal to* B if its *offset* is equal to the *offset* of B, and A is *after* B if its *offset* is greater than the *offset* of B.

In the second case a child C of the *container* [p.265] of A is an *ancestor container* [p.265] of B. In this case, A is *before* B if the *offset* [p.265] of A is less than or equal to the index of the child C and A is *after* B otherwise.

In the third case a child C of the *container* [p.265] of B is an *ancestor container* [p.265] of A. In this case, A is *before* B if the index of the child C is less than the *offset* [p.265] of B and A is *after* B otherwise.

In the fourth case, none of three other cases hold: the containers of A and B are siblings or descendants of sibling nodes. In this case, A is*before* B if the *container* [p.265] of A is before the *container* of B in a pre-order traversal of the Ranges’ *context tree* [p.266] and A is *after* B otherwise.

Note that because the same location in a text representation of the document can correspond to two different positions in the DOM tree, it is possible for two boundary-points to not compare equal even though they would be equal in the text representation. For this reason, the informal definition above can sometimes be incorrect.

8.6. Deleting Content with a Range

One can delete the contents selected by a Range with:

```
  void deleteContents();
```

`deleteContents()` deletes all nodes and characters selected by the Range. All other nodes and characters remain in the *context tree* [p.266] of the Range. Some examples of this deletion operation are:

```
(1) <FOO>AB<MOO>CD</MOO>CD</FOO>  -->
<FOO>A^CD</FOO>

(2) <FOO>A<MOO>BC</MOO>DE</FOO>  -->
<FOO>A<MOO>B</MOO>^E</FOO>

(3) <FOO>XY<BAR>ZW</BAR>Q</FOO>  -->
<FOO>X^<BAR>W</BAR>Q</FOO>

(4) <FOO><BAR1>AB</BAR1><BAR2/><BAR3>CD</BAR3></FOO>
-->  <FOO><BAR1>A</BAR1>^<BAR3>D</BAR3>
```

After `deleteContents()` is invoked on a Range, the Range is *collapsed* [p.267] . If no node was *partially selected* [p.267] by the Range, then it is *collapsed* to its original start point, as in example (1). If a node was *partially selected* by the Range and was an *ancestor container* [p.265] of the start of the Range and no ancestor of the node satisfies these two conditions, then the Range is collapsed to the position immediately after the node, as in examples (2) and (4). If a node was *partially selected* by the Range and was an *ancestor container* of the end of the Range and no ancestor of the node satisfies these two conditions, then the Range is collapsed to the position immediately before the node, as in examples (3) and (4).

Note that if deletion of a Range leaves adjacent Text nodes, they are not automatically merged, and empty Text nodes are not automatically removed. Two Text nodes should be joined only if each is the container of one of the boundary-points of a Range whose contents are deleted. To merge adjacent Text nodes, or remove empty text nodes, the `normalize()` method on the `Node` [p.38] interface should be used.

8.7. Extracting Content

If the contents of a Range need to be extracted rather than deleted, the following method may be used:

```
  DocumentFragment extractContents();
```

The `extractContents()` method removes nodes from the Range’s *context tree* [p.266] similarly to the `deleteContents()` method. In addition, it places the deleted contents in a new `DocumentFragment` [p.28] . The following examples illustrate the contents of the returned DocumentFragment:

```
(1) <FOO>AB<MOO>CD</MOO>CD</FOO>  -->
B<MOO>CD</MOO>

(2) <FOO>A<MOO>BC</MOO>DE</FOO>  -->
<MOO>C<MOO>D

(3) <FOO>XY<BAR>ZW</BAR>Q</FOO>  -->
Y<BAR>Z</BAR>

(4)
<FOO><BAR1>AB</BAR1><BAR2/><BAR3>CD</BAR3></FOO> -->
<BAR1>B</BAR1><BAR2/><BAR3>C</BAR3>
```

It is important to note that nodes that are *partially selected* [p.267] by the Range are cloned. Since part of such a node’s contents must remain in the Range’s *context tree* [p.266] and part of the contents must be moved to the new DocumentFragment, a clone of the *partially selected* node is included in the new DocumentFragment. Note that cloning does not take place for *selected* [p.267] elements; these nodes are moved to the new DocumentFragment.

8.8. Cloning Content

The contents of a Range may be duplicated using the following method:

```
  DocumentFragment cloneContents();
```

This method returns a `DocumentFragment` [p.28] that is similar to the one returned by the method `extractContents()`. However, in this case, the original nodes and character data in the Range are not removed from the Range's *context tree* [p.266] . Instead, all of the nodes and text content within the returned `DocumentFragment`are cloned.

8.9. Inserting Content

A node may be inserted into a Range using the following method:

```
  void insertNode(in Node n) raises(RangeException);
```

The `insertNode()` method inserts the specified node into the Range's *context tree* [p.266] . The node is inserted at the start *boundary-point* [p.265] of the Range, without modifying it.

If the start boundary point of the Range is in a `Text` [p.66] node, the `insertNode` operation splits the `Text` node at the boundary point. If the node to be inserted is also a `Text` node, the resulting adjacent `Text` nodes are not normalized automatically; this operation is left to the application.

The Node passed into this method can be a `DocumentFragment` [p.28] . In that case, the contents of the `DocumentFragment` are inserted at the start *boundary-point* [p.265] of the Range, but the `DocumentFragment` itself is not. Note that if the Node represents the root of a sub-tree, the entire sub-tree is inserted.

The same rules that apply to the `insertBefore()` method on the Node interface apply here. Specifically, the Node passed in, if it already has a parent, will be removed from its existing position.

8.10. Surrounding Content

The insertion of a single node to subsume the content selected by a Range can be performed with:

```
  void surroundContents(in Node n);
```

The `surroundContents()` method causes all of the content selected by the Range to be rooted by the specified node. Calling `surroundContents()` with the node FOO in the following examples yields:

```
     Before:
       <BAR>AB<MOO>C</MOO>DE</BAR>
     After surroundContents(FOO):

<BAR>A<FOO>B<MOO>C</MOO>D</FOO>E</BAR>
```

Another way of describing the effect of this method on the Range's *context tree* [p.266] is to decompose it in terms of other operations:

1. Remove the contents selected by the Range with a call to `extractContents()`.
2. Insert `node` where the Range is now collapsed (after the extraction) with `insertNode().`
3. Insert the entire contents of the extracted DocumentFragment into `node`. Specifically, invoke the `appendChild()` on `node` passing in the DocumentFragment returned as a result of the call to `extractContents()`
4. Select `node` and all of its contents with `selectNode()`.

The `surroundContents()` method raises an exception if the Range *partially selects* [p.267] a non-Text node. An example of a Range for which `surroundContents()`raises an exception is:

<FOO>A**B<BAR>C**D</BAR>E</FOO>

If `node` has any children, those children are removed before its insertion. Also, if `node` already has a parent, it is removed from the original parent's `childNodes` list.

8.11. Miscellaneous Members

One can clone a Range:

```
Range cloneRange();
```

This creates a new Range which selects exactly the same content as that selected by the Range on which the method `cloneRange` was invoked. No content is affected by this operation.

Because the boundary-points of a Range do not necessarily have the same *container* [p.265] s, use:

```
readonly attribute Node commonAncestorContainer;
```

to get the *ancestor container* [p.265] of both boundary-points that is furthest down from the Range's *root container* [p.266]

One can get a copy of all the character data selected or partially selected by a Range with:

```
DOMString toString();
```

This does nothing more than simply concatenate all the character data selected by the Range. This includes character data in both `Text` [p.66] and `CDATASection` [p.67] nodes.

8.12. Range modification under document mutation

As a document is modified, the Ranges within the document need to be updated. For example, if one boundary-point of a Range is within a node and that node is removed from the document, then the Range would be invalid unless it is fixed up in some way. This section describes how Ranges are modified under document mutations so that they remain valid.

There are two general principles which apply to Ranges under document mutation: The first is that all Ranges in a document will remain valid after any mutation operation and the second is that, as much as possible, all Ranges will select the same portion of the document after any mutation operation.

Any mutation of the document tree which affect Ranges can be considered to be a combination of basic deletion and insertion operations. In fact, it can be convenient to think of those operations as being accomplished using the `deleteContents()` and `insertNode()` Range methods and, in the case of Text mutations, the `splitText()` and `normalize()` methods.

8.12.1. Insertions

An insertion occurs at a single point, the insertion point, in the document. For any Range in the document tree, consider each boundary-point. The only case in which the boundary-point will be changed after the insertion is when the boundary-point and the insertion point have the same *container* [p.265] and the *offset* [p.265] of the insertion point is strictly less than the *offset* of the Range's boundary-point. In that case the *offset* of the Range's boundary-point will be increased so that it is between the same nodes or characters as it was before the insertion.

Note that when content is inserted at a boundary-point, it is ambiguous as to where the boundary-point should be repositioned if its relative position is to be maintained. There are two possibilities: at the start or at the end of the newly inserted content. We have chosen that in this case neither the *container* [p.265] nor *offset* [p.265] of the boundary-point is changed. As a result, the boundary-point will be positioned at the start of the newly inserted content.

Examples:

Suppose the Range selects the following:

<P>Abcd efgh X**Y blah i**jkl</P>

Consider the insertion of the text "*inserted text*" at the following positions:

1. Before the 'X':

<P>Abcd efgh *inserted text*X**Y blah i**jkl</P>

2. After the 'X':

<P>Abcd efgh X***inserted text*****Y blah i**jkl</P>

3. After the 'Y':

<P>Abcd efgh X**Y*inserted text* blah i**jkl</P>

4. After the 'h' in "Y blah":

<P>Abcd efgh X**Y blah*inserted text* i**jkl</P>

8.12.2. Deletions

Any deletion from the document tree can be considered as a sequence of `deleteContents()` operations applied to a minimal set of disjoint Ranges. To specify how a Range is modified under deletions we need only consider what happens to a Range under a single `deleteContents()`operation of another Range. And, in fact, we need only consider what happens to a single boundary-point of the Range since both boundary-points are modified using the same algorithm.

If a boundary-point of the original Range is within the content being deleted, then after the deletion it will be at the same position as the resulting boundary-point of the (now *collapsed* [p.267]) Range used to delete the contents.

If a boundary-point is after the content being deleted then it is not affected by the deletion unless its *container* [p.265] is also the *container* of one of the boundary-points of the Range being deleted. If there is such a common *container*, then the index of the boundary-point is modified so that the boundary-point maintains its position relative to the content of the *container*.

If a boundary-point is before the content being deleted then it is not affected by the deletion at all.

Examples:

In these examples, the Range on which `deleteContents()`is invoked is indicated by the underline.

Example 1.

Before:

```
<P>Abcd efgh The Range ijkl</P>
```

After:

```
<P>Abcd Range ijkl</P>
```

Example 2.

Before:

```
<p>Abcd efgh The Range ijkl</p>
```

After:

```
<p>Abcd ^kl</p>
```

Example 3.

Before:

```
<P>ABCD efgh The <EM>Range</EM> ijkl</P>
```

After:

`<P>ABCD <EM>`**`ange`**`</EM> ijkl</P>`

In this example, the container of the start boundary-point after the deletion is the Text node holding the string "ange".

Example 4.

Before:

`<P>Abcd efgh T`**`he Range i`**`jkl</P>`

After:

`<P>Abcd `**`he Range i`**`jkl</P>`

Example 5.

Before:

`<P>Abcd <EM>efgh T`**`he Range i`**`j</EM>kl</P>`

After:

`<P>Abcd ^kl</P>`

8.13. Formal Description of the Range Interface

To summarize, the complete, formal description of the `Range` [p.276] interface is given below:

Interface *Range* (introduced in **DOM Level 2**)

IDL Definition

```
// Introduced in DOM Level 2:
interface Range {
  readonly attribute Node             startContainer;
                                        // raises(DOMException) on retrieval

  readonly attribute long             startOffset;
                                        // raises(DOMException) on retrieval

  readonly attribute Node             endContainer;
                                        // raises(DOMException) on retrieval

  readonly attribute long             endOffset;
                                        // raises(DOMException) on retrieval

  readonly attribute boolean          collapsed;
                                        // raises(DOMException) on retrieval

  readonly attribute Node             commonAncestorContainer;
                                        // raises(DOMException) on retrieval
  void               setStart(in Node refNode,
                              in long offset)
                                        raises(RangeException,
                                               DOMException);
  void               setEnd(in Node refNode,
                            in long offset)
                                        raises(RangeException,
                                               DOMException);
  void               setStartBefore(in Node refNode)
                                        raises(RangeException,
                                               DOMException);
  void               setStartAfter(in Node refNode)
                                        raises(RangeException,
                                               DOMException);
  void               setEndBefore(in Node refNode)
                                        raises(RangeException,
                                               DOMException);
  void               setEndAfter(in Node refNode)
                                        raises(RangeException,
                                               DOMException);
  void               collapse(in boolean toStart)
                                        raises(DOMException);
  void               selectNode(in Node refNode)
                                        raises(RangeException,
                                               DOMException);
  void               selectNodeContents(in Node refNode)
                                        raises(RangeException,
                                               DOMException);
  // CompareHow
  const unsigned short      START_TO_START                 = 0;
  const unsigned short      START_TO_END                   = 1;
  const unsigned short      END_TO_END                     = 2;
  const unsigned short      END_TO_START                   = 3;

  short              compareBoundaryPoints(in unsigned short how,
                                           in Range sourceRange)
                                        raises(DOMException);
  void               deleteContents()
                                        raises(DOMException);
  DocumentFragment   extractContents()
                                        raises(DOMException);
  DocumentFragment   cloneContents()
                                        raises(DOMException);
  void               insertNode(in Node newNode)
                                        raises(DOMException,
                                               RangeException);
  void               surroundContents(in Node newParent)
                                        raises(DOMException,
                                               RangeException);
  Range              cloneRange()
                                        raises(DOMException);
  DOMString          toString()
                                        raises(DOMException);
  void               detach()
                                        raises(DOMException);
};
```

Definition group *CompareHow*

Passed as a parameter to the `compareBoundaryPoints` method.

Defined Constants

`END_TO_END`
: Compare end boundary-point of `sourceRange` to end boundary-point of Range on which `compareBoundaryPoints` is invoked.

`END_TO_START`
: Compare end boundary-point of `sourceRange` to start boundary-point of Range on which `compareBoundaryPoints` is invoked.

`START_TO_END`
: Compare start boundary-point of `sourceRange` to end boundary-point of Range on which `compareBoundaryPoints` is invoked.

`START_TO_START`
: Compare start boundary-point of `sourceRange` to start boundary-point of Range on which `compareBoundaryPoints` is invoked.

Attributes

`collapsed` of type `boolean`, readonly

TRUE if the Range is collapsed

Exceptions on retrieval

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

`commonAncestorContainer` of type `Node` [p.38] , readonly

The *deepest* [p.448] common *ancestor container* [p.265] of the Range’s two boundary-points.

Exceptions on retrieval

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

`endContainer` of type `Node` [p.38] , readonly

Node within which the Range ends

Exceptions on retrieval

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

`endOffset` of type `long`, readonly
Offset within the ending node of the Range.
Exceptions on retrieval

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

`startContainer` of type `Node` [p.38] , readonly
Node within which the Range begins
Exceptions on retrieval

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

`startOffset` of type `long`, readonly
Offset within the starting node of the Range.
Exceptions on retrieval

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

Methods

`cloneContents`
Duplicates the contents of a Range
Return Value

`DocumentFragment` [p.28]	A DocumentFragment that contains content equivalent to this Range.

Exceptions

`DOMException` [p.24]	HIERARCHY_REQUEST_ERR: Raised if a DocumentType node would be extracted into the new DocumentFragment. INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Parameters

`cloneRange`
Produces a new Range whose boundary-points are equal to the boundary-points of the Range.
Return Value

`Range` [p.276]	The duplicated Range.

Exceptions

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Parameters

`collapse`

Collapse a Range onto one of its boundary-points

Parameters

`toStart` of type `boolean`

If TRUE, collapses the Range onto its start; if FALSE, collapses it onto its end.

Exceptions

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`compareBoundaryPoints`

Compare the boundary-points of two Ranges in a document.

Parameters

`how` of type `unsigned short`

`sourceRange` of type `Range` [p.276]

Return Value

`short`	-1, 0 or 1 depending on whether the corresponding boundary-point of the Range is before, equal to, or after the corresponding boundary-point of `sourceRange`.

Exceptions

`DOMException` [p.24]	WRONG_DOCUMENT_ERR: Raised if the two Ranges are not in the same Document or DocumentFragment. INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

`deleteContents`

Removes the contents of a Range from the containing document or document fragment without returning a reference to the removed content.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if any portion of the content of the Range is read-only or any of the nodes that contain any of the content of the Range are read-only. INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Parameters
No Return Value

`detach`

Called to indicate that the Range is no longer in use and that the implementation may relinquish any resources associated with this Range. Subsequent calls to any methods or attribute getters on this Range will result in a `DOMException` [p.24] being thrown with an error code of `INVALID_STATE_ERR`.

Exceptions

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Parameters
No Return Value

`extractContents`

Moves the contents of a Range from the containing document or document fragment to a new DocumentFragment.

Return Value

`DocumentFragment` [p.28]	A DocumentFragment containing the extracted contents.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if any portion of the content of the Range is read-only or any of the nodes which contain any of the content of the Range are read-only. HIERARCHY_REQUEST_ERR: Raised if a DocumentType node would be extracted into the new DocumentFragment. INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Parameters

`insertNode`

Inserts a node into the Document or DocumentFragment at the start of the Range. If the container is a Text node, this will be split at the start of the Range. Adjacent Text nodes will not be automatically merged.

Parameters

`newNode` of type `Node` [p.38]

The node to insert at the start of the Range

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if an *ancestor container* [p.265] of the start of the Range is read-only. WRONG_DOCUMENT_ERR: Raised if `newNode` and the *container* [p.265] of the start of the Range were not created from the same document. HIERARCHY_REQUEST_ERR: Raised if the *container* [p.265] of the start of the Range is of a type that does not allow children of the type of `newNode` or if `newNode` is an ancestor of the *container*. INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.
`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if `node` is an Attr, Entity, Notation, DocumentFragment, or Document node.

No Return Value

`selectNode`

Select a node and its contents

Parameters

`refNode` of type `Node` [p.38]

The node to select.

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if an ancestor of `refNode` is an Entity, Notation or DocumentType node or if `refNode` is a Document, DocumentFragment, Attr, Entity, or Notation node.
`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`selectNodeContents`

Select the contents within a node

Parameters

`refNode` of type `Node` [p.38]

Node to select from

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if `refNode` or an ancestor of `refNode` is an Entity, Notation or DocumentType node.
`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`setEnd`

Sets the attributes describing the end of a Range.

Parameters

`refNode` of type `Node` [p.38]

The `refNode` value. This parameter must be different from `null`.

`offset` of type `long`

The `endOffset` value.

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if `refNode` or an ancestor of `refNode` is an Entity, Notation, or DocumentType node.
`DOMException` [p.24]	INDEX_SIZE_ERR: Raised if `offset` is negative or greater than the number of child units in `refNode`. Child units are 16-bit units if `refNode` is a CharacterData, Comment or ProcessingInstruction node. Child units are Nodes in all other cases.
	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`setEndAfter`

Sets the end of a Range to be after a node

Parameters

`refNode` of type `Node` [p.38]

Range ends after `refNode`.

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if the root container of `refNode`is not an Attr, Document or DocumentFragment node or if `refNode` is a Document, DocumentFragment, Attr, Entity, or Notation node.
`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`setEndBefore`

Sets the end position to be before a node.

Parameters

`refNode` of type `Node` [p.38]

Range ends before `refNode`

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if the root container of `refNode`is not an Attr, Document, or DocumentFragment node or if `refNode`is a Document, DocumentFragment, Attr, Entity, or Notation node.
`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`setStart`

Sets the attributes describing the start of the Range.

Parameters

`refNode` of type `Node` [p.38]

The `refNode` value. This parameter must be different from `null`.

`offset` of type `long`

The `startOffset` value.

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if `refNode` or an ancestor of `refNode` is an Entity, Notation, or DocumentType node.
`DOMException` [p.24]	INDEX_SIZE_ERR: Raised if `offset` is negative or greater than the number of child units in `refNode`. Child units are 16-bit units if `refNode` is a CharacterData, Comment or ProcessingInstruction node. Child units are Nodes in all other cases.
	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`setStartAfter`

Sets the start position to be after a node

Parameters

`refNode` of type `Node` [p.38]

Range starts after `refNode`

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if the root container of `refNode`is not an Attr, Document, or DocumentFragment node or if `refNode`is a Document, DocumentFragment, Attr, Entity, or Notation node.
`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`setStartBefore`

Sets the start position to be before a node

Parameters

`refNode` of type `Node` [p.38]

Range starts before `refNode`

Exceptions

`RangeException` [p.288]	INVALID_NODE_TYPE_ERR: Raised if the root container of `refNode`is not an Attr, Document, or DocumentFragment node or if `refNode`is a Document, DocumentFragment, Attr, Entity, or Notation node.
`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Return Value

`surroundContents`

Reparents the contents of the Range to the given node and inserts the node at the position of the start of the Range.

Parameters

`newParent` of type `Node` [p.38]

The node to surround the contents with.

Exceptions

`DOMException` [p.24]	NO_MODIFICATION_ALLOWED_ERR: Raised if an *ancestor container* [p.265] of either boundary-point of the Range is read-only. WRONG_DOCUMENT_ERR: Raised if `newParent` and the *container* [p.265] of the start of the Range were not created from the same document. HIERARCHY_REQUEST_ERR: Raised if the *container* [p.265] of the start of the Range is of a type that does not allow children of the type of `newParent` or if `newParent` is an ancestor of the *container*or if `node` would end up with a child node of a type not allowed by the type of `node`. INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.
`RangeException` [p.288]	BAD_BOUNDARYPOINTS_ERR: Raised if the Range *partially selects* [p.267] a non-text node. INVALID_NODE_TYPE_ERR: Raised if `node` is an Attr, Entity, DocumentType, Notation, Document, or DocumentFragment node.

No Return Value

`toString`

Returns the contents of a Range as a string. This string contains only the data characters, not any markup.

Return Value

`DOMString` [p.21]	The contents of the Range. **Issue data:** Only the data characters. **Resolution:**

Exceptions

`DOMException` [p.24]	INVALID_STATE_ERR: Raised if `detach()` has already been invoked on this object.

No Parameters

Interface *DocumentRange* (introduced in **DOM Level 2**)

IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentRange {
  Range              createRange();
};
```

Methods

`createRange`

This interface can be obtained from the object implementing the `Document` [p.29] interface using binding-specific casting methods.

Return Value

`Range` [p.276]	The initial state of the Range returned from this method is such that both of its boundary-points are positioned at the beginning of the corresponding Document, before any content. The Range returned can only be used to select content associated with this Document, or with DocumentFragments and Attrs for which this Document is the `ownerDocument`.

No Parameters

No Exceptions

Exception *RangeException* introduced in **DOM Level 2**

Range operations may throw a `RangeException` [p.288] as specified in their method descriptions.

IDL Definition

```
// Introduced in DOM Level 2:
exception RangeException {
  unsigned short   code;
};

// RangeExceptionCode
const unsigned short      BAD_BOUNDARYPOINTS_ERR         = 1;
const unsigned short      INVALID_NODE_TYPE_ERR          = 2;
```

Definition group *RangeExceptionCode*

An integer indicating the type of error generated.

Defined Constants

`BAD_BOUNDARYPOINTS_ERR`

If the boundary-points of a Range do not meet specific requirements.

`INVALID_NODE_TYPE_ERR`

If the *container* [p.265] of an boundary-point of a Range is being set to either a node of an invalid type or a node with an ancestor of an invalid type.

Appendix A: Changes

Editors
Arnaud Le Hors, W3C
Philippe Le Hégaret, W3C

A.1: Changes between DOM Level 1 and DOM Level 2

A.1.1: Changes to DOM Level 1 interfaces and exceptions

Interface Attr [p.56]
The `Attr` [p.56] interface has one new attribute: `ownerElement`.

Interface Document [p.29]
The `Document` [p.29] interface has five new methods: `importNode`, `createElementNS`, `createAttributeNS`, `getElementsByTagNameNS` and `getElementById`.

Interface NamedNodeMap [p.48]
The `NamedNodeMap` [p.48] interface has three new methods: `getNamedItemNS`, `setNamedItemNS`, `removeNamedItemNS`.

Interface Node [p.38]
The `Node` [p.38] interface has two new methods: `supports` and `normalize`.
The `Node` [p.38] interface has three new attributes: `namespaceURI`, `prefix` and `localName`.
The `ownerDocument` attribute was specified to be `null` when the node is a `Document` [p.29] . It now is also `null` when the node is a `DocumentType` [p.68] which is not used with any `Document` yet.

Interface DocumentType [p.68]
The `DocumentType` [p.68] interface has three attributes: `publicId`, `systemId` and `internalSubset`.

Interface DOMImplementation [p.26]
The `DOMImplementation` [p.26] interface has two new methods: `createDocumentType` and `createDocument`.

Interface Element [p.57]
The `Element` [p.57] interface has eight new methods: `getAttributeNS`, `setAttributeNS`, `removeAttributeNS`, `getAttributeNodeNS`, `setAttributeNodeNS`, `getElementsByTagNameNS`, `hasAttribute` and `hasAttributeNS`.
The method `normalize` is now inherited from the `Node` [p.38] interface where it was moved.

Exception DOMException [p.24]
The `DOMException` [p.24] has five new exception codes: `INVALID_STATE_ERR`, `SYNTAX_ERR`, `INVALID_MODIFICATION_ERR`, `NAMESPACE_ERR` and `INVALID_ACCESS_ERR`.

A.1.2: New features

A.1.2.1: New types

`DOMTimeStamp` [p.22]
: The `DOMTimeStamp` [p.22] type was added to the Core module.

A.1.2.2: New interfaces

HTML [p.73]
: The `HTMLDOMImplementation` [p.74] interface was added to the HTML module.
On the `HTMLDocument` [p.76] interface, the method `getElementById` is now inherited from the `Document` [p.29] interface where it was moved.
On the `HTMLFrameElement` [p.121] , `HTMLIFrameElement` [p.122] , and `HTMLObjectElement` [p.106] interfaces the attribute `contentDocument` was added.

Views [p.125]
: This new module defines the interfaces `AbstractView` [p.125] and `DocumentView` [p.126] .

StyleSheets [p.127]
: This new module defines the following interfaces: `StyleSheet` [p.127] , `StyleSheetList` [p.128] , `MediaList` [p.129] , `DocumentStyle` [p.131] and `LinkStyle` [p.131] .

CSS [p.133]
: This new module defines the following interfaces: `CSS2Azimuth` [p.163] , `CSS2BackgroundPosition` [p.165] , `CSS2BorderSpacing` [p.168] , `CSS2CounterIncrement` [p.171] , `CSS2CounterReset` [p.170] , `CSS2Cursor` [p.172] , `CSS2FontFaceSrc` [p.176] , `CSS2FontFaceWidths` [p.177] , `CSS2PageSize` [p.178] , `CSS2PlayDuring` [p.173] , `CSS2Properties` [p.181] , `CSS2TextShadow` [p.174] , `CSSCharsetRule` [p.141] , `CSSFontFaceRule` [p.140] , `CSSImportRule` [p.141] , `CSSMediaRule` [p.138] , `CSSPageRule` [p.140] , `CSSPrimitiveValue` [p.147] , `CSSRule` [p.136] , `CSSRuleList` [p.135] , `CSSStyleDeclaration` [p.142] , `CSSStyleRule` [p.138] , `CSSStyleSheet` [p.134] , `CSSUnknownRule` [p.142] , `CSSValue` [p.146] , `CSSValueList` [p.154] , `Counter` [p.155] , `RGBColor` [p.154] , `Rect` [p.155] , `ViewCSS` [p.156] , `DocumentCSS` [p.157] , `DOMImplementationCSS` [p.158] and `ElementCSSInlineStyle` [p.158] .

Events [p.221]
: This new module defines the following interfaces: `Event` [p.227] , `EventListener` [p.226] , `EventTarget` [p.224] , `DocumentEvent` [p.230] , `MutationEvent` [p.238] , `UIEvent` [p.231] and `MouseEvent` [p.233] , and the exception `EventException` [p.229] .

Traversal [p.243]
: This new module defines the following interfaces: `NodeFilter` [p.255] , `NodeIterator` [p.253] , `TreeWalker` [p.258] , and `DocumentTraversal` [p.261] .

Range [p.265]
: This new module defines the interfaces `Range` [p.276] , `DocumentRange` [p.288] and the exception `RangeException` [p.288] .

Appendix B: Accessing code point boundaries

Mark Davis, IBM
Lauren Wood, SoftQuad Software Inc.

B.1: Introduction

This appendix is an informative, not a normative, part of the Level 2 DOM specification.

Characters are represented in Unicode by numbers called *code points* (also called *scalar values*). These numbers can range from 0 up to 1,114,111 = $10FFFF_{16}$ (although some of these values are illegal). Each code point can be directly encoded with a 32-bit code unit. This encoding is termed UCS-4 (or UTF-32). The DOM specification, however, uses UTF-16, in which the most frequent characters (which have values less than $FFFF_{16}$) are represented by a single 16-bit code unit, while characters above $FFFF_{16}$ use a special pair of code units called a *surrogate pair*. For more information, see [Unicode] or the Unicode Web site.

While indexing by code points as opposed to code units is not common in programs, some specifications such as XPath (and therefore XSLT and XPointer) use code point indices. For interfacing with such formats it is recommended that the programming language provide string processing methods for converting code point indices to code unit indices and back. Some languages do not provide these functions natively; for these it is recommended that the native `String` type that is bound to `DOMString` [p.21] be extended to enable this conversion. An example of how such an API might look is supplied below.

Note: Since these methods are supplied as an illustrative example of the type of functionality that is required, the names of the methods, exceptions, and interface may differ from those given here.

B.2: Methods

Interface *StringExtend*

Extensions to a language’s native String class or interface
IDL Definition

```
interface StringExtend {
  int                findOffset16(in int offset32)
                                        raises(StringIndexOutOfBoundsException);
  int                findOffset32(in int offset16)
                                        raises(StringIndexOutOfBoundsException);
};
```

Methods

`findOffset16`

Returns the UTF-16 offset that corresponds to a UTF-32 offset. Used for random access.

Note: You can always roundtrip from a UTF-32 offset to a UTF-16 offset and back. You can roundtrip from a UTF-16 offset to a UTF-32 offset and back if and only if the offset16 is not in the middle of a surrogate pair. Unmatched surrogates count as a single UTF-16 value.

Parameters
`offset32` of type `int`
UTF-32 offset.

Return Value

`int`	UTF-16 offset

Exceptions

`StringIndexOutOfBoundsException`	if `offset32` is out of bounds.

`findOffset32`

Returns the UTF-32 offset corresponding to a UTF-16 offset. Used for random access. To find the UTF-32 length of a string, use:

```
len32 = findOffset32(source, source.length());
```

Note: If the UTF-16 offset is into the middle of a surrogate pair, then the UTF-32 offset of the *end* of the pair is returned; that is, the index of the char after the end of the pair. You can always roundtrip from a UTF-32 offset to a UTF-16 offset and back. You can roundtrip from a UTF-16 offset to a UTF-32 offset and back if and only if the offset16 is not in the middle of a surrogate pair. Unmatched surrogates count as a single UTF-16 value.

Parameters
`offset16` of type `int`
UTF-16 offset

Return Value

`int`	UTF-32 offset

Exceptions

`StringIndexOutOfBoundsException`	if offset16 is out of bounds.

Appendix C: IDL Definitions

This appendix contains the complete OMG IDL for the Level 2 Document Object Model definitions. The definitions are divided into Core [p.293] , HTML [p.298] , Stylesheets [p.308] , CSS [p.309] , Events [p.323] , TreeWalkers, Filters, and Iterators [p.325] , and Range [p.327] .

The IDL files are also available as: http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510/idl.zip

C.1: Document Object Model Core

dom.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: dom.idl
#ifndef _DOM_IDL_
#define _DOM_IDL_

#pragma prefix "w3c.org"
module dom
{
  typedef sequence<unsigned short> DOMString;

  typedef   unsigned long long DOMTimeStamp;

  interface DocumentType;
  interface Document;
  interface NodeList;
  interface NamedNodeMap;
  interface Element;

  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;
  // Introduced in DOM Level 2:
  const unsigned short      INVALID_STATE_ERR              = 11;
  // Introduced in DOM Level 2:
  const unsigned short      SYNTAX_ERR                     = 12;
  // Introduced in DOM Level 2:
  const unsigned short      INVALID_MODIFICATION_ERR       = 13;
  // Introduced in DOM Level 2:
  const unsigned short      NAMESPACE_ERR                  = 14;
```

```
// Introduced in DOM Level 2:
const unsigned short      INVALID_ACCESS_ERR             = 15;


interface DOMImplementation {
  boolean            hasFeature(in DOMString feature,
                                in DOMString version);
  // Introduced in DOM Level 2:
  DocumentType       createDocumentType(in DOMString qualifiedName,
                                        in DOMString publicId,
                                        in DOMString systemId)
                                      raises(DOMException);
  // Introduced in DOM Level 2:
  Document           createDocument(in DOMString namespaceURI,
                                    in DOMString qualifiedName,
                                    in DocumentType doctype)
                                      raises(DOMException);
};

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;
  // Modified in DOM Level 2:
  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);
    // Introduced in DOM Level 2:
    void                normalize();
    // Introduced in DOM Level 2:
    boolean             supports(in DOMString feature,
                                 in DOMString version);
    // Introduced in DOM Level 2:
    readonly attribute DOMString        namespaceURI;
    // Introduced in DOM Level 2:
             attribute DOMString        prefix;
                                        // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute DOMString        localName;
  };

  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;
    // Introduced in DOM Level 2:
    Node                getNamedItemNS(in DOMString namespaceURI,
                                       in DOMString localName);
    // Introduced in DOM Level 2:
    Node                setNamedItemNS(in Node arg)
                                          raises(DOMException);
    // Introduced in DOM Level 2:
    Node                removeNamedItemNS(in DOMString namespaceURI,
                                           in DOMString localName)
                                          raises(DOMException);
  };

  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;
             attribute DOMString        value;
                                        // raises(DOMException) on setting

    // Introduced in DOM Level 2:
    readonly attribute Element          ownerElement;
  };

  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);
    // Introduced in DOM Level 2:
    DOMString          getAttributeNS(in DOMString namespaceURI,
                                      in DOMString localName);
    // Introduced in DOM Level 2:
    void               setAttributeNS(in DOMString namespaceURI,
                                      in DOMString qualifiedName,
                                      in DOMString value)
                                        raises(DOMException);
    // Introduced in DOM Level 2:
    void               removeAttributeNS(in DOMString namespaceURI,
                                         in DOMString localName)
                                        raises(DOMException);
    // Introduced in DOM Level 2:
    Attr               getAttributeNodeNS(in DOMString namespaceURI,
                                          in DOMString localName);
    // Introduced in DOM Level 2:
    Attr               setAttributeNodeNS(in Attr newAttr)
                                        raises(DOMException);
    // Introduced in DOM Level 2:
    NodeList           getElementsByTagNameNS(in DOMString namespaceURI,
                                              in DOMString localName);
    // Introduced in DOM Level 2:
    boolean            hasAttribute(in DOMString name);
    // Introduced in DOM Level 2:
    boolean            hasAttributeNS(in DOMString namespaceURI,
                                      in DOMString localName);
  };

  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;
    // Introduced in DOM Level 2:
    readonly attribute DOMString        publicId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString        systemId;
    // Introduced in DOM Level 2:
    readonly attribute DOMString        internalSubset;
  };

  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);
    // Introduced in DOM Level 2:
    Node               importNode(in Node importedNode,
                                 in boolean deep)
                                          raises(DOMException);
    // Introduced in DOM Level 2:
    Element            createElementNS(in DOMString namespaceURI,
                                       in DOMString qualifiedName)
                                          raises(DOMException);
    // Introduced in DOM Level 2:
    Attr               createAttributeNS(in DOMString namespaceURI,
                                          in DOMString qualifiedName)
                                          raises(DOMException);
    // Introduced in DOM Level 2:
    NodeList           getElementsByTagNameNS(in DOMString namespaceURI,
                                                in DOMString localName);
    // Introduced in DOM Level 2:
    Element            getElementById(in DOMString elementId);
  };
};

#endif // _DOM_IDL_
```

C.2: Document Object Model HTML

html.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// 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::DOMImplementation DOMImplementation;
  typedef dom::Document Document;
  typedef dom::NodeList NodeList;
  typedef dom::Element Element;

  interface HTMLDocument;
  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);
  };

  // Introduced in DOM Level 2:
  interface HTMLDOMImplementation : DOMImplementation {
    HTMLDocument       createHTMLDocument(in DOMString title);
  };

  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);
    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;
  // Introduced in DOM Level 2:
  readonly attribute Document          contentDocument;
};

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;
  // Introduced in DOM Level 2:
  readonly attribute Document         contentDocument;
};

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;
  // Introduced in DOM Level 2:
    readonly attribute Document         contentDocument;
  };
};

#endif // _HTML_IDL_
```

C.3: Document Object Model Views

views.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: views.idl
#ifndef _VIEWS_IDL_
#define _VIEWS_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module views
{
  interface DocumentView;

  // Introduced in DOM Level 2:
  interface AbstractView {
    readonly attribute DocumentView     document;
  };

  // Introduced in DOM Level 2:
  interface DocumentView {
    readonly attribute AbstractView     defaultView;
  };
};

#endif // _VIEWS_IDL_
```

C.4: Document Object Model StyleSheets

stylesheets.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: stylesheets.idl
#ifndef _STYLESHEETS_IDL_
#define _STYLESHEETS_IDL_

#include "dom.idl"
#include "html.idl"

#pragma prefix "dom.w3c.org"
module stylesheets
{
  typedef dom::DOMString DOMString;
  typedef dom::Node Node;
  interface MediaList;

  // Introduced in DOM Level 2:
  interface StyleSheet {
    readonly attribute DOMString        type;
             attribute boolean          disabled;
    readonly attribute Node             ownerNode;
    readonly attribute StyleSheet       parentStyleSheet;
    readonly attribute DOMString        href;
    readonly attribute DOMString        title;
    readonly attribute MediaList        media;
  };

  // Introduced in DOM Level 2:
  interface StyleSheetList {
    readonly attribute unsigned long    length;
    StyleSheet         item(in unsigned long index);
  };

  // Introduced in DOM Level 2:
  interface MediaList {
             attribute DOMString        mediaText;
                                        // raises(dom::DOMException) on setting

    readonly attribute unsigned long    length;
    DOMString          item(in unsigned long index);
    void               deleteMedium(in DOMString oldMedium)
                                        raises(dom::DOMException);
    void               appendMedium(in DOMString newMedium)
                                        raises(dom::DOMException);
  };

  // Introduced in DOM Level 2:
  interface LinkStyle {
    readonly attribute StyleSheet       sheet;
  };

  // Introduced in DOM Level 2:
  interface DocumentStyle {
    readonly attribute StyleSheetList   styleSheets;
  };
};

#endif // _STYLESHEETS_IDL_
```

C.5: Document Object Model CSS

css.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: css.idl
#ifndef _CSS_IDL_
#define _CSS_IDL_

#include "dom.idl"
```

```
#include "stylesheets.idl"
#include "html.idl"
#include "views.idl"

#pragma prefix "dom.w3c.org"
module css
{
  typedef dom::DOMString DOMString;
  typedef dom::Element Element;
  typedef dom::DOMImplementation DOMImplementation;

  interface CSSRule;
  interface CSSStyleSheet;
  interface CSSStyleDeclaration;
  interface CSSValue;
  interface Counter;
  interface Rect;
  interface RGBColor;

  // Introduced in DOM Level 2:
  interface CSSRuleList {
    readonly attribute unsigned long    length;
    CSSRule            item(in unsigned long index);
  };

  // Introduced in DOM Level 2:
  interface CSSRule {
    // RuleType
    const unsigned short      UNKNOWN_RULE                   = 0;
    const unsigned short      STYLE_RULE                     = 1;
    const unsigned short      CHARSET_RULE                   = 2;
    const unsigned short      IMPORT_RULE                    = 3;
    const unsigned short      MEDIA_RULE                     = 4;
    const unsigned short      FONT_FACE_RULE                 = 5;
    const unsigned short      PAGE_RULE                      = 6;

    readonly attribute unsigned short   type;
             attribute DOMString        cssText;
                                        // raises(dom::DOMException) on setting

    readonly attribute CSSStyleSheet    parentStyleSheet;
    readonly attribute CSSRule          parentRule;
  };

  // Introduced in DOM Level 2:
  interface CSSStyleRule : CSSRule {
             attribute DOMString        selectorText;
                                        // raises(dom::DOMException) on setting

    readonly attribute CSSStyleDeclaration  style;
  };

  // Introduced in DOM Level 2:
  interface CSSMediaRule : CSSRule {
    readonly attribute stylesheets::MediaList  media;
    readonly attribute CSSRuleList      cssRules;
    unsigned long      insertRule(in DOMString rule,
                                  in unsigned long index)
                                        raises(dom::DOMException);
    void                deleteRule(in unsigned long index)
                                        raises(dom::DOMException);
  };

  // Introduced in DOM Level 2:
  interface CSSFontFaceRule : CSSRule {
    readonly attribute CSSStyleDeclaration  style;
  };

  // Introduced in DOM Level 2:
  interface CSSPageRule : CSSRule {
             attribute DOMString        selectorText;
                                        // raises(dom::DOMException) on setting

    readonly attribute CSSStyleDeclaration  style;
  };

  // Introduced in DOM Level 2:
  interface CSSImportRule : CSSRule {
    readonly attribute DOMString        href;
    readonly attribute stylesheets::MediaList  media;
    readonly attribute CSSStyleSheet    styleSheet;
  };

  // Introduced in DOM Level 2:
  interface CSSCharsetRule : CSSRule {
             attribute DOMString        encoding;
                                        // raises(dom::DOMException) on setting

  };

  // Introduced in DOM Level 2:
  interface CSSUnknownRule : CSSRule {
  };

  // Introduced in DOM Level 2:
  interface CSSStyleDeclaration {
             attribute DOMString        cssText;
                                        // raises(dom::DOMException) on setting

    DOMString          getPropertyValue(in DOMString propertyName);
    CSSValue           getPropertyCSSValue(in DOMString propertyName);
    DOMString          removeProperty(in DOMString propertyName)
                                        raises(dom::DOMException);
    DOMString          getPropertyPriority(in DOMString propertyName);
    void               setProperty(in DOMString propertyName,
                                   in DOMString value,
                                   in DOMString priority)
                                        raises(dom::DOMException);
    readonly attribute unsigned long    length;
    DOMString          item(in unsigned long index);
    readonly attribute CSSRule          parentRule;
  };

  // Introduced in DOM Level 2:
```

```
interface CSSValue {
  // UnitTypes
  const unsigned short      CSS_INHERIT                    = 0;
  const unsigned short      CSS_PRIMITIVE_VALUE            = 1;
  const unsigned short      CSS_VALUE_LIST                 = 2;
  const unsigned short      CSS_CUSTOM                     = 3;

           attribute DOMString        cssText;
                                      // raises(dom::DOMException) on setting

  readonly attribute unsigned short   valueType;
};

// Introduced in DOM Level 2:
interface CSSPrimitiveValue : CSSValue {
  // UnitTypes
  const unsigned short      CSS_UNKNOWN                    = 0;
  const unsigned short      CSS_NUMBER                     = 1;
  const unsigned short      CSS_PERCENTAGE                 = 2;
  const unsigned short      CSS_EMS                        = 3;
  const unsigned short      CSS_EXS                        = 4;
  const unsigned short      CSS_PX                         = 5;
  const unsigned short      CSS_CM                         = 6;
  const unsigned short      CSS_MM                         = 7;
  const unsigned short      CSS_IN                         = 8;
  const unsigned short      CSS_PT                         = 9;
  const unsigned short      CSS_PC                         = 10;
  const unsigned short      CSS_DEG                        = 11;
  const unsigned short      CSS_RAD                        = 12;
  const unsigned short      CSS_GRAD                       = 13;
  const unsigned short      CSS_MS                         = 14;
  const unsigned short      CSS_S                          = 15;
  const unsigned short      CSS_HZ                         = 16;
  const unsigned short      CSS_KHZ                        = 17;
  const unsigned short      CSS_DIMENSION                  = 18;
  const unsigned short      CSS_STRING                     = 19;
  const unsigned short      CSS_URI                        = 20;
  const unsigned short      CSS_IDENT                      = 21;
  const unsigned short      CSS_ATTR                       = 22;
  const unsigned short      CSS_COUNTER                    = 23;
  const unsigned short      CSS_RECT                       = 24;
  const unsigned short      CSS_RGBCOLOR                   = 25;

  readonly attribute unsigned short   primitiveType;
  void               setFloatValue(in unsigned short unitType,
                                  in float floatValue)
                                      raises(dom::DOMException);
  float              getFloatValue(in unsigned short unitType)
                                      raises(dom::DOMException);
  void               setStringValue(in unsigned short stringType,
                                   in DOMString stringValue)
                                      raises(dom::DOMException);
  DOMString          getStringValue()
                                      raises(dom::DOMException);
  Counter            getCounterValue()
                                      raises(dom::DOMException);
  Rect               getRectValue()
```

```
                                      raises(dom::DOMException);
  RGBColor           getRGBColorValue()
                                      raises(dom::DOMException);
};

// Introduced in DOM Level 2:
interface CSSValueList : CSSValue {
  readonly attribute unsigned long    length;
  CSSValue           item(in unsigned long index);
};

// Introduced in DOM Level 2:
interface RGBColor {
  readonly attribute CSSPrimitiveValue  red;
  readonly attribute CSSPrimitiveValue  green;
  readonly attribute CSSPrimitiveValue  blue;
};

// Introduced in DOM Level 2:
interface Rect {
  readonly attribute CSSPrimitiveValue  top;
  readonly attribute CSSPrimitiveValue  right;
  readonly attribute CSSPrimitiveValue  bottom;
  readonly attribute CSSPrimitiveValue  left;
};

// Introduced in DOM Level 2:
interface Counter {
  readonly attribute DOMString        identifier;
  readonly attribute DOMString        listStyle;
  readonly attribute DOMString        separator;
};

// Introduced in DOM Level 2:
interface ElementCSSInlineStyle {
  readonly attribute CSSStyleDeclaration  style;
};

// Introduced in DOM Level 2:
interface CSS2Azimuth : CSSValue {
  readonly attribute unsigned short   azimuthType;
  readonly attribute DOMString        identifier;
  readonly attribute boolean          behind;
  void               setAngleValue(in unsigned short uType,
                                   in float fValue)
                                      raises(dom::DOMException);
  float              getAngleValue(in unsigned short uType)
                                      raises(dom::DOMException);
  void               setIdentifier(in DOMString ident,
                                   in boolean b)
                                      raises(dom::DOMException);
};

// Introduced in DOM Level 2:
interface CSS2BackgroundPosition : CSSValue {
  readonly attribute unsigned short   horizontalType;
  readonly attribute unsigned short   verticalType;
  readonly attribute DOMString        horizontalIdentifier;
  readonly attribute DOMString        verticalIdentifier;
  float              getHorizontalPosition(in float hType)
                                      raises(dom::DOMException);
  float              getVerticalPosition(in float vType)
                                      raises(dom::DOMException);
  void               setHorizontalPosition(in unsigned short hType, 
                                           in float value)
                                      raises(dom::DOMException);
  void               setVerticalPosition(in unsigned short vType, 
                                         in float value)
                                      raises(dom::DOMException);
  void               setPositionIdentifier(in DOMString hIdentifier, 
                                           in DOMString vIdentifier)
                                      raises(dom::DOMException);
};

// Introduced in DOM Level 2:
interface CSS2BorderSpacing : CSSValue {
  readonly attribute unsigned short   horizontalType;
  readonly attribute unsigned short   verticalType;
  float              getHorizontalSpacing(in float hType)
                                      raises(dom::DOMException);
  float              getVerticalSpacing(in float vType)
                                      raises(dom::DOMException);
  void               setHorizontalSpacing(in unsigned short hType, 
                                          in float value)
                                      raises(dom::DOMException);
  void               setVerticalSpacing(in unsigned short vType, 
                                        in float value)
                                      raises(dom::DOMException);
};

// Introduced in DOM Level   2:
interface CSS2CounterReset : CSSValue {
           attribute DOMString        identifier;
                                      // raises(dom::DOMException) on setting

           attribute short            reset;
                                      // raises(dom::DOMException) on setting

};

// Introduced in DOM   Level 2:
interface CSS2CounterIncrement : CSSValue {
           attribute DOMString        identifier;
                                      // raises(dom::DOMException) on setting

           attribute short            increment;
                                      // raises(dom::DOMException) on setting

};

// Introduced in DOM Level 2:
interface CSS2Cursor : CSSValue {
  readonly attribute CSSValueList     uris;
           attribute DOMString        predefinedCursor;
```

```
                                    // raises(dom::DOMException) on setting

};

// Introduced in DOM Level 2:
interface CSS2PlayDuring : CSSValue {
  readonly attribute unsigned short   playDuringType;
           attribute DOMString        playDuringIdentifier;
                                    // raises(dom::DOMException) on setting

           attribute DOMString        uri;
                                    // raises(dom::DOMException) on setting

           attribute boolean          mix;
                                    // raises(dom::DOMException) on setting

           attribute boolean          repeat;
                                    // raises(dom::DOMException) on setting

};

// Introduced in DOM Level 2:
interface CSS2TextShadow : CSSValue {
  readonly attribute CSSValue         color;
  readonly attribute CSSValue         horizontal;
  readonly attribute CSSValue         vertical;
  readonly attribute CSSValue         blur;
};

// Introduced in DOM Level 2:
interface CSS2FontFaceSrc : CSSValue {
           attribute DOMString        uri;
                                    // raises(dom::DOMException) on setting

  readonly attribute CSSValueList     format;
           attribute DOMString        fontFaceName;
                                    // raises(dom::DOMException) on setting

};

// Introduced in DOM Level 2:
interface CSS2FontFaceWidths : CSSValue {
           attribute DOMString        urange;
                                    // raises(dom::DOMException) on setting

  readonly attribute CSSValueList     numbers;
};

// Introduced in DOM Level 2:
interface CSS2PageSize : CSSValue {
  readonly attribute unsigned short   widthType;
  readonly attribute unsigned short   heightType;
  readonly attribute DOMString        identifier;
  float              getWidth(in float wType)
                                    raises(dom::DOMException);
  float              getHeightSize(in float hType)
                                    raises(dom::DOMException);
  void                setWidthSize(in unsigned short wType,
                                   in float value)
                                       raises(dom::DOMException);
  void                setHeightSize(in unsigned short hType,
                                    in float value)
                                       raises(dom::DOMException);
  void                setIdentifier(in DOMString ident)
                                       raises(dom::DOMException);
};

// Introduced in DOM Level 2:
interface CSS2Properties {
           attribute DOMString        azimuth;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        background;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        backgroundAttachment;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        backgroundColor;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        backgroundImage;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        backgroundPosition;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        backgroundRepeat;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        border;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderCollapse;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderColor;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderSpacing;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderStyle;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderTop;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderRight;
                                       // raises(dom::DOMException) on setting

           attribute DOMString        borderBottom;
                                       // raises(dom::DOMException) on setting
```

```
attribute DOMString         borderLeft;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderTopColor;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderRightColor;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderBottomColor;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderLeftColor;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderTopStyle;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderRightStyle;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderBottomStyle;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderLeftStyle;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderTopWidth;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderRightWidth;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderBottomWidth;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderLeftWidth;
                            // raises(dom::DOMException) on setting

attribute DOMString         borderWidth;
                            // raises(dom::DOMException) on setting

attribute DOMString         bottom;
                            // raises(dom::DOMException) on setting

attribute DOMString         captionSide;
                            // raises(dom::DOMException) on setting

attribute DOMString         clear;
                            // raises(dom::DOMException) on setting

attribute DOMString         clip;
                            // raises(dom::DOMException) on setting

attribute DOMString         color;
                            // raises(dom::DOMException) on setting
```

```
attribute DOMString        content;
                           // raises(dom::DOMException) on setting

attribute DOMString        counterIncrement;
                           // raises(dom::DOMException) on setting

attribute DOMString        counterReset;
                           // raises(dom::DOMException) on setting

attribute DOMString        cue;
                           // raises(dom::DOMException) on setting

attribute DOMString        cueAfter;
                           // raises(dom::DOMException) on setting

attribute DOMString        cueBefore;
                           // raises(dom::DOMException) on setting

attribute DOMString        cursor;
                           // raises(dom::DOMException) on setting

attribute DOMString        direction;
                           // raises(dom::DOMException) on setting

attribute DOMString        display;
                           // raises(dom::DOMException) on setting

attribute DOMString        elevation;
                           // raises(dom::DOMException) on setting

attribute DOMString        emptyCells;
                           // raises(dom::DOMException) on setting

attribute DOMString        cssFloat;
                           // raises(dom::DOMException) on setting

attribute DOMString        font;
                           // raises(dom::DOMException) on setting

attribute DOMString        fontFamily;
                           // raises(dom::DOMException) on setting

attribute DOMString        fontSize;
                           // raises(dom::DOMException) on setting

attribute DOMString        fontSizeAdjust;
                           // raises(dom::DOMException) on setting

attribute DOMString        fontStretch;
                           // raises(dom::DOMException) on setting

attribute DOMString        fontStyle;
                           // raises(dom::DOMException) on setting

attribute DOMString        fontVariant;
                           // raises(dom::DOMException) on setting
```

```
attribute DOMString        fontWeight;
                           // raises(dom::DOMException) on setting

attribute DOMString        height;
                           // raises(dom::DOMException) on setting

attribute DOMString        left;
                           // raises(dom::DOMException) on setting

attribute DOMString        letterSpacing;
                           // raises(dom::DOMException) on setting

attribute DOMString        lineHeight;
                           // raises(dom::DOMException) on setting

attribute DOMString        listStyle;
                           // raises(dom::DOMException) on setting

attribute DOMString        listStyleImage;
                           // raises(dom::DOMException) on setting

attribute DOMString        listStylePosition;
                           // raises(dom::DOMException) on setting

attribute DOMString        listStyleType;
                           // raises(dom::DOMException) on setting

attribute DOMString        margin;
                           // raises(dom::DOMException) on setting

attribute DOMString        marginTop;
                           // raises(dom::DOMException) on setting

attribute DOMString        marginRight;
                           // raises(dom::DOMException) on setting

attribute DOMString        marginBottom;
                           // raises(dom::DOMException) on setting

attribute DOMString        marginLeft;
                           // raises(dom::DOMException) on setting

attribute DOMString        markerOffset;
                           // raises(dom::DOMException) on setting

attribute DOMString        marks;
                           // raises(dom::DOMException) on setting

attribute DOMString        maxHeight;
                           // raises(dom::DOMException) on setting

attribute DOMString        maxWidth;
                           // raises(dom::DOMException) on setting

attribute DOMString        minHeight;
                           // raises(dom::DOMException) on setting
```

```
attribute DOMString        minWidth;
                           // raises(dom::DOMException) on setting

attribute DOMString        orphans;
                           // raises(dom::DOMException) on setting

attribute DOMString        outline;
                           // raises(dom::DOMException) on setting

attribute DOMString        outlineColor;
                           // raises(dom::DOMException) on setting

attribute DOMString        outlineStyle;
                           // raises(dom::DOMException) on setting

attribute DOMString        outlineWidth;
                           // raises(dom::DOMException) on setting

attribute DOMString        overflow;
                           // raises(dom::DOMException) on setting

attribute DOMString        padding;
                           // raises(dom::DOMException) on setting

attribute DOMString        paddingTop;
                           // raises(dom::DOMException) on setting

attribute DOMString        paddingRight;
                           // raises(dom::DOMException) on setting

attribute DOMString        paddingBottom;
                           // raises(dom::DOMException) on setting

attribute DOMString        paddingLeft;
                           // raises(dom::DOMException) on setting

attribute DOMString        page;
                           // raises(dom::DOMException) on setting

attribute DOMString        pageBreakAfter;
                           // raises(dom::DOMException) on setting

attribute DOMString        pageBreakBefore;
                           // raises(dom::DOMException) on setting

attribute DOMString        pageBreakInside;
                           // raises(dom::DOMException) on setting

attribute DOMString        pause;
                           // raises(dom::DOMException) on setting

attribute DOMString        pauseAfter;
                           // raises(dom::DOMException) on setting

attribute DOMString        pauseBefore;
                           // raises(dom::DOMException) on setting
```

```
attribute DOMString        pitch;
                           // raises(dom::DOMException) on setting

attribute DOMString        pitchRange;
                           // raises(dom::DOMException) on setting

attribute DOMString        playDuring;
                           // raises(dom::DOMException) on setting

attribute DOMString        position;
                           // raises(dom::DOMException) on setting

attribute DOMString        quotes;
                           // raises(dom::DOMException) on setting

attribute DOMString        richness;
                           // raises(dom::DOMException) on setting

attribute DOMString        right;
                           // raises(dom::DOMException) on setting

attribute DOMString        size;
                           // raises(dom::DOMException) on setting

attribute DOMString        speak;
                           // raises(dom::DOMException) on setting

attribute DOMString        speakHeader;
                           // raises(dom::DOMException) on setting

attribute DOMString        speakNumeral;
                           // raises(dom::DOMException) on setting

attribute DOMString        speakPunctuation;
                           // raises(dom::DOMException) on setting

attribute DOMString        speechRate;
                           // raises(dom::DOMException) on setting

attribute DOMString        stress;
                           // raises(dom::DOMException) on setting

attribute DOMString        tableLayout;
                           // raises(dom::DOMException) on setting

attribute DOMString        textAlign;
                           // raises(dom::DOMException) on setting

attribute DOMString        textDecoration;
                           // raises(dom::DOMException) on setting

attribute DOMString        textIndent;
                           // raises(dom::DOMException) on setting

attribute DOMString        textShadow;
                           // raises(dom::DOMException) on setting
```

```
          attribute DOMString        textTransform;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        top;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        unicodeBidi;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        verticalAlign;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        visibility;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        voiceFamily;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        volume;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        whiteSpace;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        widows;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        width;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        wordSpacing;
                                     // raises(dom::DOMException) on setting

          attribute DOMString        zIndex;
                                     // raises(dom::DOMException) on setting

};

// Introduced in DOM Level 2:
interface CSSStyleSheet : stylesheets::StyleSheet {
  readonly attribute CSSRule          ownerRule;
  readonly attribute CSSRuleList      cssRules;
  unsigned long      insertRule(in DOMString rule,
                                in unsigned long index)
                                     raises(dom::DOMException);
  void               deleteRule(in unsigned long index)
                                     raises(dom::DOMException);
};

// Introduced in DOM Level 2:
interface ViewCSS : views::AbstractView {
  CSSStyleDeclaration getComputedStyle(in Element elt,
                                      in DOMString pseudoElt);
};

// Introduced in DOM Level 2:
interface DocumentCSS : stylesheets::DocumentStyle {
    CSSStyleDeclaration getOverrideStyle(in Element elt,
                                         in DOMString pseudoElt);
  };

  // Introduced in DOM   Level 2:
  interface DOMImplementationCSS : DOMImplementation {
    CSSStyleSheet      createCSSStyleSheet(in DOMString title,
                                           in DOMString media);
  };
};

#endif // _CSS_IDL_
```

C.6: Document Object Model Events

events.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: events.idl
#ifndef _EVENTS_IDL_
#define _EVENTS_IDL_

#include "dom.idl"
#include "views.idl"

#pragma prefix "dom.w3c.org"
module events
{
  typedef dom::DOMString DOMString;
  typedef dom::DOMTimeStamp DOMTimeStamp;
  typedef dom::Node Node;

  interface EventListener;
  interface Event;

  // Introduced in DOM Level 2:
  exception EventException {
    unsigned short   code;
  };

  // EventExceptionCode
  const unsigned short      UNSPECIFIED_EVENT_TYPE_ERR     = 0;


  // Introduced in DOM Level 2:
  interface EventTarget {
    void               addEventListener(in DOMString type,
                                        in EventListener listener,
                                        in boolean useCapture);
    void               removeEventListener(in DOMString type,
                                           in EventListener listener,
                                           in boolean useCapture);
    boolean            dispatchEvent(in Event evt)
                                        raises(EventException);
  };
```

```
// Introduced in DOM Level 2:
interface EventListener {
  void               handleEvent(in Event evt);
};

// Introduced in DOM Level 2:
interface Event {
  // PhaseType
  const unsigned short      CAPTURING_PHASE                = 1;
  const unsigned short      AT_TARGET                      = 2;
  const unsigned short      BUBBLING_PHASE                 = 3;

  readonly attribute DOMString        type;
  readonly attribute EventTarget      target;
  readonly attribute EventTarget      currentTarget;
  readonly attribute unsigned short   eventPhase;
  readonly attribute boolean          bubbles;
  readonly attribute boolean          cancelable;
  readonly attribute DOMTimeStamp     timeStamp;
  void               stopPropagation();
  void               preventDefault();
  void               initEvent(in DOMString eventTypeArg,
                               in boolean canBubbleArg,
                               in boolean cancelableArg);
};

// Introduced in DOM Level 2:
interface DocumentEvent {
  Event              createEvent(in DOMString eventType)
                                      raises(dom::DOMException);
};

// Introduced in DOM Level 2:
interface UIEvent : Event {
  readonly attribute views::AbstractView  view;
  readonly attribute long             detail;
  void               initUIEvent(in DOMString typeArg,
                                 in boolean canBubbleArg,
                                 in boolean cancelableArg,
                                 in views::AbstractView viewArg,
                                 in long detailArg);
};

// Introduced in DOM Level 2:
interface MouseEvent : UIEvent {
  readonly attribute long             screenX;
  readonly attribute long             screenY;
  readonly attribute long             clientX;
  readonly attribute long             clientY;
  readonly attribute boolean          ctrlKey;
  readonly attribute boolean          shiftKey;
  readonly attribute boolean          altKey;
  readonly attribute boolean          metaKey;
  readonly attribute unsigned short   button;
  readonly attribute EventTarget      relatedTarget;
  void               initMouseEvent(in DOMString typeArg,
                                          in boolean canBubbleArg,
                                          in boolean cancelableArg,
                                          in views::AbstractView viewArg,
                                          in long detailArg,
                                          in long screenXArg,
                                          in long screenYArg,
                                          in long clientXArg,
                                          in long clientYArg,
                                          in boolean ctrlKeyArg,
                                          in boolean altKeyArg,
                                          in boolean shiftKeyArg,
                                          in boolean metaKeyArg,
                                          in unsigned short buttonArg,
                                          in EventTarget relatedTargetArg);
  };

  // Introduced in DOM Level 2:
  interface MutationEvent : Event {
    readonly attribute Node             relatedNode;
    readonly attribute DOMString        prevValue;
    readonly attribute DOMString        newValue;
    readonly attribute DOMString        attrName;
    void               initMutationEvent(in DOMString typeArg,
                                         in boolean canBubbleArg,
                                         in boolean cancelableArg,
                                         in Node relatedNodeArg,
                                         in DOMString prevValueArg,
                                         in DOMString newValueArg,
                                         in DOMString attrNameArg);
  };
};

#endif // _EVENTS_IDL_
```

C.7: Document Object Model Traversal

traversal.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: traversal.idl
#ifndef _TRAVERSAL_IDL_
#define _TRAVERSAL_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module traversal
{
  typedef dom::Node Node;

  interface NodeFilter;

  // Introduced in DOM Level 2:
  interface NodeIterator {
    readonly attribute Node             root;
    readonly attribute unsigned long    whatToShow;
    readonly attribute NodeFilter       filter;
    readonly attribute boolean          expandEntityReferences;
    Node               nextNode()
                                        raises(dom::DOMException);
    Node               previousNode()
                                        raises(dom::DOMException);
    void               detach();
  };

  // Introduced in DOM Level 2:
  interface NodeFilter {
    // Constants returned by acceptNode
    const short              FILTER_ACCEPT                  = 1;
    const short              FILTER_REJECT                  = 2;
    const short              FILTER_SKIP                    = 3;

    // Constants for whatToShow
    const unsigned long      SHOW_ALL                       = 0xFFFFFFFF;
    const unsigned long      SHOW_ELEMENT                   = 0x00000001;
    const unsigned long      SHOW_ATTRIBUTE                 = 0x00000002;
    const unsigned long      SHOW_TEXT                      = 0x00000004;
    const unsigned long      SHOW_CDATA_SECTION             = 0x00000008;
    const unsigned long      SHOW_ENTITY_REFERENCE          = 0x00000010;
    const unsigned long      SHOW_ENTITY                    = 0x00000020;
    const unsigned long      SHOW_PROCESSING_INSTRUCTION    = 0x00000040;
    const unsigned long      SHOW_COMMENT                   = 0x00000080;
    const unsigned long      SHOW_DOCUMENT                  = 0x00000100;
    const unsigned long      SHOW_DOCUMENT_TYPE             = 0x00000200;
    const unsigned long      SHOW_DOCUMENT_FRAGMENT         = 0x00000400;
    const unsigned long      SHOW_NOTATION                  = 0x00000800;

    short              acceptNode(in Node n);
  };

  // Introduced in DOM Level 2:
  interface TreeWalker {
    readonly attribute Node             root;
    readonly attribute unsigned long    whatToShow;
    readonly attribute NodeFilter       filter;
    readonly attribute boolean          expandEntityReferences;
             attribute Node             currentNode;
                                        // raises(dom::DOMException) on setting

    Node               parentNode();
    Node               firstChild();
    Node               lastChild();
    Node               previousSibling();
    Node               nextSibling();
    Node               previousNode();
    Node               nextNode();
  };

  // Introduced in DOM Level 2:
  interface DocumentTraversal {
    NodeIterator       createNodeIterator(in Node root,
                                          in unsigned long whatToShow,
                                          in NodeFilter filter,
                                          in boolean entityReferenceExpansion)
                                        raises(dom::DOMException);
    TreeWalker         createTreeWalker(in Node root,
                                        in unsigned long whatToShow,
                                        in NodeFilter filter,
                                        in boolean entityReferenceExpansion)
                                        raises(dom::DOMException);
  };
};

#endif // _TRAVERSAL_IDL_
```

C.8: Document Object Model Range

range.idl:

```
// See also http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510
// File: range.idl
#ifndef _RANGE_IDL_
#define _RANGE_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module range
{
  typedef dom::Node Node;
  typedef dom::DocumentFragment DocumentFragment;
  typedef dom::DOMString DOMString;

  // Introduced in DOM Level 2:
  exception RangeException {
    unsigned short   code;
  };

  // RangeExceptionCode
  const unsigned short      BAD_BOUNDARYPOINTS_ERR         = 1;
  const unsigned short      INVALID_NODE_TYPE_ERR          = 2;


  // Introduced in DOM Level 2:
  interface Range {
    readonly attribute Node             startContainer;
                                        // raises(dom::DOMException) on retrieval

    readonly attribute long             startOffset;
                                        // raises(dom::DOMException) on retrieval

    readonly attribute Node             endContainer;
                                        // raises(dom::DOMException) on retrieval

    readonly attribute long             endOffset;
                                        // raises(dom::DOMException) on retrieval
```

```
readonly attribute boolean          collapsed;
                                    // raises(dom::DOMException) on retrieval

readonly attribute Node             commonAncestorContainer;
                                    // raises(dom::DOMException) on retrieval

void               setStart(in Node refNode,
                            in long offset)
                                    raises(RangeException,
                                           dom::DOMException);
void               setEnd(in Node refNode,
                          in long offset)
                                    raises(RangeException,
                                           dom::DOMException);
void               setStartBefore(in Node refNode)
                                    raises(RangeException,
                                           dom::DOMException);
void               setStartAfter(in Node refNode)
                                    raises(RangeException,
                                           dom::DOMException);
void               setEndBefore(in Node refNode)
                                    raises(RangeException,
                                           dom::DOMException);
void               setEndAfter(in Node refNode)
                                    raises(RangeException,
                                           dom::DOMException);
void               collapse(in boolean toStart)
                                    raises(dom::DOMException);
void               selectNode(in Node refNode)
                                    raises(RangeException,
                                           dom::DOMException);
void               selectNodeContents(in Node refNode)
                                    raises(RangeException,
                                           dom::DOMException);
// CompareHow
const unsigned short      START_TO_START                 = 0;
const unsigned short      START_TO_END                   = 1;
const unsigned short      END_TO_END                     = 2;
const unsigned short      END_TO_START                   = 3;

short              compareBoundaryPoints(in unsigned short how,
                                         in Range sourceRange)
                                    raises(dom::DOMException);
void               deleteContents()
                                    raises(dom::DOMException);
DocumentFragment   extractContents()
                                    raises(dom::DOMException);
DocumentFragment   cloneContents()
                                    raises(dom::DOMException);
void               insertNode(in Node newNode)
                                    raises(dom::DOMException,
                                           RangeException);
void               surroundContents(in Node newParent)
                                    raises(dom::DOMException,
                                           RangeException);
Range              cloneRange()
                                    raises(dom::DOMException);
```

```
    DOMString          toString()
                                        raises(dom::DOMException);
    void               detach()
                                        raises(dom::DOMException);
  };

  // Introduced in DOM Level 2:
  interface DocumentRange {
    Range              createRange();
  };
};

#endif // _RANGE_IDL_
```

range.idl:

Appendix D: Java Language Binding

This appendix contains the complete Java bindings for the Level 2 Document Object Model. The definitions are divided into Core [p.331] , HTML [p.339] , StyleSheets [p.364] , CSS [p.365] , Events [p.386] , Filters and Iterators [p.389] , and Range [p.391] .

The Java files are also available as
http://www.w3.org/TR/2000/CR-DOM-Level-2-20000510/java-binding.zip

D.1: Document Object Model 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;
    /**
     * @since DOM Level 2
     */
    public static final short INVALID_STATE_ERR         = 11;
    /**
     * @since DOM Level 2
     */
    public static final short SYNTAX_ERR                = 12;
    /**
     * @since DOM Level 2
     */
    public static final short INVALID_MODIFICATION_ERR  = 13;
    /**
     * @since DOM Level 2
     */
    public static final short NAMESPACE_ERR             = 14;
    /**
     * @since DOM Level 2
     */
    public static final short INVALID_ACCESS_ERR        = 15;

}
```

org/w3c/dom/DOMImplementation.java:

```
package org.w3c.dom;

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

    public DocumentType createDocumentType(String qualifiedName,
                                           String publicId,
                                           String systemId)
                                           throws DOMException;

    public Document createDocument(String namespaceURI,
                                   String qualifiedName,
                                   DocumentType doctype)
                                   throws DOMException;

}
```

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);

    public Node importNode(Node importedNode,
                           boolean deep)
                           throws DOMException;

    public Element createElementNS(String namespaceURI,
                                   String qualifiedName)
                                   throws DOMException;

    public Attr createAttributeNS(String namespaceURI,
                                  String qualifiedName)
                                  throws DOMException;

    public NodeList getElementsByTagNameNS(String namespaceURI,
                                           String localName);

    public Element getElementById(String elementId);

}
```

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);

    public void normalize();

    public boolean supports(String feature,
                            String version);

    public String getNamespaceURI();

    public String getPrefix();
    public void setPrefix(String prefix)
                            throws DOMException;

    public String getLocalName();

}
```

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();

    public Node getNamedItemNS(String namespaceURI,
                               String localName);

    public Node setNamedItemNS(Node arg)
                               throws DOMException;

    public Node removeNamedItemNS(String namespaceURI,
                                  String localName)
                                  throws DOMException;

}
```

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;

    public Element getOwnerElement();

}
```

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 String getAttributeNS(String namespaceURI,
                             String localName);

    public void setAttributeNS(String namespaceURI,
                           String qualifiedName,
                           String value)
                           throws DOMException;

    public void removeAttributeNS(String namespaceURI,
                              String localName)
                              throws DOMException;

    public Attr getAttributeNodeNS(String namespaceURI,
                               String localName);

    public Attr setAttributeNodeNS(Attr newAttr)
                               throws DOMException;

    public NodeList getElementsByTagNameNS(String namespaceURI,
                                       String localName);

    public boolean hasAttribute(String name);

    public boolean hasAttributeNS(String namespaceURI,
                              String localName);

}
```

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();

    public String getPublicId();

    public String getSystemId();

    public String getInternalSubset();

}
```

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;

}
```

D.2: Document Object Model HTML

org/w3c/dom/html/HTMLDOMImplementation.java:

```
package org.w3c.dom.html;

import org.w3c.dom.DOMImplementation;

public interface HTMLDOMImplementation extends DOMImplementation {
    public HTMLDocument createHTMLDocument(String title);

}
```

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;

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 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:

```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:

```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:

```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:

```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:

```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:

```java
package org.w3c.dom.html;

import org.w3c.dom.Document;

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);

    public Document getContentDocument();

}
```

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;

import org.w3c.dom.Document;

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);

    public Document getContentDocument();

}
```

org/w3c/dom/html/HTMLIFrameElement.java:

```
package org.w3c.dom.html;

import org.w3c.dom.Document;

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);

    public Document getContentDocument();

}
```

D.3: Document Object Model Views

org/w3c/dom/views/AbstractView.java:

```
package org.w3c.dom.views;

public interface AbstractView {
    public DocumentView getDocument();

}
```

org/w3c/dom/views/DocumentView.java:

```
package org.w3c.dom.views;

public interface DocumentView {
    public AbstractView getDefaultView();

}
```

D.4: Document Object Model StyleSheets

org/w3c/dom/stylesheets/StyleSheet.java:

```
package org.w3c.dom.stylesheets;

import org.w3c.dom.Node;

public interface StyleSheet {
    public String getType();

    public boolean getDisabled();
    public void setDisabled(boolean disabled);

    public Node getOwnerNode();

    public StyleSheet getParentStyleSheet();

    public String getHref();

    public String getTitle();

    public MediaList getMedia();

}
```

org/w3c/dom/stylesheets/StyleSheetList.java:

```
package org.w3c.dom.stylesheets;

public interface StyleSheetList {
    public int getLength();
    public StyleSheet item(int index);

}
```

org/w3c/dom/stylesheets/MediaList.java:

```
package org.w3c.dom.stylesheets;

import org.w3c.dom.DOMException;

public interface MediaList {
    public String getMediaText();
    public void setMediaText(String mediaText)
                           throws DOMException;

    public int getLength();

    public String item(int index);

    public void deleteMedium(String oldMedium)
                             throws DOMException;

    public void appendMedium(String newMedium)
                             throws DOMException;

}
```

org/w3c/dom/stylesheets/LinkStyle.java:

```
package org.w3c.dom.stylesheets;

public interface LinkStyle {
    public StyleSheet getSheet();

}
```

org/w3c/dom/stylesheets/DocumentStyle.java:

```
package org.w3c.dom.stylesheets;

public interface DocumentStyle {
    public StyleSheetList getStyleSheets();

}
```

D.5: Document Object Model CSS

org/w3c/dom/css/CSSStyleSheet.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;
import org.w3c.dom.stylesheets.StyleSheet;

public interface CSSStyleSheet extends StyleSheet {
    public CSSRule getOwnerRule();

    public CSSRuleList getCssRules();

    public int insertRule(String rule,
                          int index)
                          throws DOMException;

    public void deleteRule(int index)
                           throws DOMException;

}
```

org/w3c/dom/css/CSSRuleList.java:

```
package org.w3c.dom.css;

public interface CSSRuleList {
    public int getLength();

    public CSSRule item(int index);

}
```

org/w3c/dom/css/CSSRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSRule {
    // RuleType
    public static final short UNKNOWN_RULE              = 0;
    public static final short STYLE_RULE                = 1;
    public static final short CHARSET_RULE              = 2;
    public static final short IMPORT_RULE               = 3;
    public static final short MEDIA_RULE                = 4;
    public static final short FONT_FACE_RULE            = 5;
    public static final short PAGE_RULE                 = 6;

    public short getType();

    public String getCssText();
    public void setCssText(String cssText)
                        throws DOMException;

    public CSSStyleSheet getParentStyleSheet();
    public CSSRule getParentRule();

}
```

org/w3c/dom/css/CSSStyleRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSStyleRule extends CSSRule {
    public String getSelectorText();
    public void setSelectorText(String selectorText)
                        throws DOMException;

    public CSSStyleDeclaration getStyle();

}
```

org/w3c/dom/css/CSSMediaRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;
import org.w3c.dom.stylesheets.MediaList;

public interface CSSMediaRule extends CSSRule {
    public MediaList getMedia();

    public CSSRuleList getCssRules();

    public int insertRule(String rule,
                          int index)
                          throws DOMException;

    public void deleteRule(int index)
                           throws DOMException;

}
```

org/w3c/dom/css/CSSFontFaceRule.java:

```
package org.w3c.dom.css;

public interface CSSFontFaceRule extends CSSRule {
    public CSSStyleDeclaration getStyle();

}
```

org/w3c/dom/css/CSSPageRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSPageRule extends CSSRule {
    public String getSelectorText();
    public void setSelectorText(String selectorText)
                           throws DOMException;

    public CSSStyleDeclaration getStyle();

}
```

org/w3c/dom/css/CSSImportRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.stylesheets.MediaList;

public interface CSSImportRule extends CSSRule {
    public String getHref();

    public MediaList getMedia();

    public CSSStyleSheet getStyleSheet();

}
```

org/w3c/dom/css/CSSCharsetRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSCharsetRule extends CSSRule {
    public String getEncoding();
    public void setEncoding(String encoding)
                           throws DOMException;

}
```

org/w3c/dom/css/CSSUnknownRule.java:

```
package org.w3c.dom.css;

public interface CSSUnknownRule extends CSSRule {
}
```

org/w3c/dom/css/CSSStyleDeclaration.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSStyleDeclaration {
    public String getCssText();
    public void setCssText(String cssText)
                           throws DOMException;

    public String getPropertyValue(String propertyName);

    public CSSValue getPropertyCSSValue(String propertyName);

    public String removeProperty(String propertyName)
                                 throws DOMException;

    public String getPropertyPriority(String propertyName);

    public void setProperty(String propertyName,
                            String value,
                            String priority)
                            throws DOMException;

    public int getLength();

    public String item(int index);

    public CSSRule getParentRule();

}
```

org/w3c/dom/css/CSSValue.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSValue {
    // UnitTypes
    public static final short CSS_INHERIT               = 0;
    public static final short CSS_PRIMITIVE_VALUE       = 1;
    public static final short CSS_VALUE_LIST            = 2;
    public static final short CSS_CUSTOM                = 3;

    public String getCssText();
    public void setCssText(String cssText)
                       throws DOMException;

    public short getValueType();

}
```

org/w3c/dom/css/CSSPrimitiveValue.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSSPrimitiveValue extends CSSValue {
    // UnitTypes
    public static final short CSS_UNKNOWN               = 0;
    public static final short CSS_NUMBER                = 1;
    public static final short CSS_PERCENTAGE            = 2;
    public static final short CSS_EMS                   = 3;
    public static final short CSS_EXS                   = 4;
    public static final short CSS_PX                    = 5;
    public static final short CSS_CM                    = 6;
    public static final short CSS_MM                    = 7;
    public static final short CSS_IN                    = 8;
    public static final short CSS_PT                    = 9;
    public static final short CSS_PC                    = 10;
    public static final short CSS_DEG                   = 11;
    public static final short CSS_RAD                   = 12;
    public static final short CSS_GRAD                  = 13;
    public static final short CSS_MS                    = 14;
    public static final short CSS_S                     = 15;
    public static final short CSS_HZ                    = 16;
    public static final short CSS_KHZ                   = 17;
    public static final short CSS_DIMENSION             = 18;
    public static final short CSS_STRING                = 19;
    public static final short CSS_URI                   = 20;
    public static final short CSS_IDENT                 = 21;
    public static final short CSS_ATTR                  = 22;
    public static final short CSS_COUNTER               = 23;
    public static final short CSS_RECT                  = 24;
    public static final short CSS_RGBCOLOR              = 25;

    public short getPrimitiveType();

    public void setFloatValue(short unitType,
                              float floatValue)
                              throws DOMException;

    public float getFloatValue(short unitType)
                               throws DOMException;

    public void setStringValue(short stringType,
                               String stringValue)
                               throws DOMException;

    public String getStringValue()
                                 throws DOMException;

    public Counter getCounterValue()
                                   throws DOMException;

    public Rect getRectValue()
                             throws DOMException;
    public RGBColor getRGBColorValue()
                                      throws DOMException;

}
```

org/w3c/dom/css/CSSValueList.java:

```
package org.w3c.dom.css;

public interface CSSValueList extends CSSValue {
    public int getLength();

    public CSSValue item(int index);

}
```

org/w3c/dom/css/RGBColor.java:

```
package org.w3c.dom.css;

public interface RGBColor {
    public CSSPrimitiveValue getRed();

    public CSSPrimitiveValue getGreen();

    public CSSPrimitiveValue getBlue();

}
```

org/w3c/dom/css/Rect.java:

```
package org.w3c.dom.css;

public interface Rect {
    public CSSPrimitiveValue getTop();

    public CSSPrimitiveValue getRight();

    public CSSPrimitiveValue getBottom();

    public CSSPrimitiveValue getLeft();

}
```

org/w3c/dom/css/Counter.java:

```
package org.w3c.dom.css;

public interface Counter {
    public String getIdentifier();

    public String getListStyle();
    public String getSeparator();

}
```

org/w3c/dom/css/ViewCSS.java:

```
package org.w3c.dom.css;

import org.w3c.dom.Element;
import org.w3c.dom.views.AbstractView;

public interface ViewCSS extends AbstractView {
    public CSSStyleDeclaration getComputedStyle(Element elt,
                                                String pseudoElt);

}
```

org/w3c/dom/css/DocumentCSS.java:

```
package org.w3c.dom.css;

import org.w3c.dom.Element;
import org.w3c.dom.stylesheets.DocumentStyle;

public interface DocumentCSS extends DocumentStyle {
    public CSSStyleDeclaration getOverrideStyle(Element elt,
                                                String pseudoElt);

}
```

org/w3c/dom/css/DOMImplementationCSS.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMImplementation;

public interface DOMImplementationCSS extends DOMImplementation {
    public CSSStyleSheet createCSSStyleSheet(String title,
                                             String media);

}
```

org/w3c/dom/css/ElementCSSInlineStyle.java:

```
package org.w3c.dom.css;

public interface ElementCSSInlineStyle {
    public CSSStyleDeclaration getStyle();

}
```

org/w3c/dom/css/CSS2Azimuth.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2Azimuth extends CSSValue {
    public short getAzimuthType();

    public String getIdentifier();

    public boolean getBehind();

    public void setAngleValue(short uType,
                              float fValue)
                              throws DOMException;

    public float getAngleValue(short uType)
                               throws DOMException;

    public void setIdentifier(String ident,
                              boolean b)
                              throws DOMException;

}
```

org/w3c/dom/css/CSS2BackgroundPosition.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2BackgroundPosition extends CSSValue {
    public short getHorizontalType();

    public short getVerticalType();

    public String getHorizontalIdentifier();

    public String getVerticalIdentifier();

    public float getHorizontalPosition(float hType)
                                       throws DOMException;

    public float getVerticalPosition(float vType)
                                     throws DOMException;

    public void setHorizontalPosition(short hType,
                                      float value)
                                      throws DOMException;

    public void setVerticalPosition(short vType,
                                    float value)
                                    throws DOMException;

    public void setPositionIdentifier(String hIdentifier,
                                      String vIdentifier)
                                      throws DOMException;

}
```

org/w3c/dom/css/CSS2BorderSpacing.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2BorderSpacing extends CSSValue {
    public short getHorizontalType();

    public short getVerticalType();

    public float getHorizontalSpacing(float hType)
                                      throws DOMException;

    public float getVerticalSpacing(float vType)
                                    throws DOMException;

    public void setHorizontalSpacing(short hType,
                                     float value)
                                     throws DOMException;

    public void setVerticalSpacing(short vType,
                                   float value)
                                   throws DOMException;

}
```

org/w3c/dom/css/CSS2CounterReset.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2CounterReset extends CSSValue {
    public String getIdentifier();
    public void setIdentifier(String identifier)
                                   throws DOMException;

    public short getReset();
    public void setReset(short reset)
                                   throws DOMException;

}
```

org/w3c/dom/css/CSS2CounterIncrement.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2CounterIncrement extends CSSValue {
    public String getIdentifier();
    public void setIdentifier(String identifier)
                                   throws DOMException;

    public short getIncrement();
    public void setIncrement(short increment)
                                   throws DOMException;

}
```

org/w3c/dom/css/CSS2Cursor.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2Cursor extends CSSValue {
    public CSSValueList getUris();

    public String getPredefinedCursor();
    public void setPredefinedCursor(String predefinedCursor)
                                   throws DOMException;

}
```

org/w3c/dom/css/CSS2PlayDuring.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2PlayDuring extends CSSValue {
    public short getPlayDuringType();

    public String getPlayDuringIdentifier();
    public void setPlayDuringIdentifier(String playDuringIdentifier)
                                   throws DOMException;

    public String getUri();
    public void setUri(String uri)
                                   throws DOMException;

    public boolean getMix();
    public void setMix(boolean mix)
                                   throws DOMException;

    public boolean getRepeat();
    public void setRepeat(boolean repeat)
                                   throws DOMException;

}
```

org/w3c/dom/css/CSS2TextShadow.java:

```
package org.w3c.dom.css;

public interface CSS2TextShadow extends CSSValue {
    public CSSValue getColor();

    public CSSValue getHorizontal();

    public CSSValue getVertical();

    public CSSValue getBlur();

}
```

org/w3c/dom/css/CSS2FontFaceSrc.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2FontFaceSrc extends CSSValue {
    public String getUri();
    public void setUri(String uri)
                                   throws DOMException;

    public CSSValueList getFormat();

    public String getFontFaceName();
    public void setFontFaceName(String fontFaceName)
                                   throws DOMException;

}
```

org/w3c/dom/css/CSS2FontFaceWidths.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2FontFaceWidths extends CSSValue {
    public String getUrange();
    public void setUrange(String urange)
                                   throws DOMException;

    public CSSValueList getNumbers();

}
```

org/w3c/dom/css/CSS2PageSize.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2PageSize extends CSSValue {
    public short getWidthType();

    public short getHeightType();

    public String getIdentifier();

    public float getWidth(float wType)
                          throws DOMException;

    public float getHeightSize(float hType)
                               throws DOMException;

    public void setWidthSize(short wType,
                             float value)
                             throws DOMException;

    public void setHeightSize(short hType,
                              float value)
                              throws DOMException;

    public void setIdentifier(String ident)
                              throws DOMException;

}
```

org/w3c/dom/css/CSS2Properties.java:

```
package org.w3c.dom.css;

import org.w3c.dom.DOMException;

public interface CSS2Properties {
    public String getAzimuth();
    public void setAzimuth(String azimuth)
                              throws DOMException;

    public String getBackground();
    public void setBackground(String background)
                              throws DOMException;

    public String getBackgroundAttachment();
    public void setBackgroundAttachment(String backgroundAttachment)
                              throws DOMException;

    public String getBackgroundColor();
    public void setBackgroundColor(String backgroundColor)
                              throws DOMException;

    public String getBackgroundImage();
    public void setBackgroundImage(String backgroundImage)
                              throws DOMException;

    public String getBackgroundPosition();
    public void setBackgroundPosition(String backgroundPosition)
                              throws DOMException;

    public String getBackgroundRepeat();
    public void setBackgroundRepeat(String backgroundRepeat)
                              throws DOMException;

    public String getBorder();
    public void setBorder(String border)
                              throws DOMException;

    public String getBorderCollapse();
    public void setBorderCollapse(String borderCollapse)
                              throws DOMException;

    public String getBorderColor();
    public void setBorderColor(String borderColor)
                              throws DOMException;

    public String getBorderSpacing();
    public void setBorderSpacing(String borderSpacing)
                              throws DOMException;

    public String getBorderStyle();
    public void setBorderStyle(String borderStyle)
                              throws DOMException;

    public String getBorderTop();
    public void setBorderTop(String borderTop)
                              throws DOMException;

    public String getBorderRight();
    public void setBorderRight(String borderRight)
                              throws DOMException;

    public String getBorderBottom();
    public void setBorderBottom(String borderBottom)
                              throws DOMException;

    public String getBorderLeft();
    public void setBorderLeft(String borderLeft)
                              throws DOMException;

    public String getBorderTopColor();
    public void setBorderTopColor(String borderTopColor)
                              throws DOMException;

    public String getBorderRightColor();
    public void setBorderRightColor(String borderRightColor)
                              throws DOMException;

    public String getBorderBottomColor();
    public void setBorderBottomColor(String borderBottomColor)
```

```
                          throws DOMException;

    public String getBorderLeftColor();
    public void setBorderLeftColor(String borderLeftColor)
                          throws DOMException;

    public String getBorderTopStyle();
    public void setBorderTopStyle(String borderTopStyle)
                          throws DOMException;

    public String getBorderRightStyle();
    public void setBorderRightStyle(String borderRightStyle)
                          throws DOMException;

    public String getBorderBottomStyle();
    public void setBorderBottomStyle(String borderBottomStyle)
                          throws DOMException;

    public String getBorderLeftStyle();
    public void setBorderLeftStyle(String borderLeftStyle)
                          throws DOMException;

    public String getBorderTopWidth();
    public void setBorderTopWidth(String borderTopWidth)
                          throws DOMException;

    public String getBorderRightWidth();
    public void setBorderRightWidth(String borderRightWidth)
                          throws DOMException;

    public String getBorderBottomWidth();
    public void setBorderBottomWidth(String borderBottomWidth)
                          throws DOMException;

    public String getBorderLeftWidth();
    public void setBorderLeftWidth(String borderLeftWidth)
                          throws DOMException;

    public String getBorderWidth();
    public void setBorderWidth(String borderWidth)
                          throws DOMException;

    public String getBottom();
    public void setBottom(String bottom)
                          throws DOMException;

    public String getCaptionSide();
    public void setCaptionSide(String captionSide)
                          throws DOMException;

    public String getClear();
    public void setClear(String clear)
                          throws DOMException;

    public String getClip();
    public void setClip(String clip)
                          throws DOMException;
```

```
public String getColor();
public void setColor(String color)
                          throws DOMException;

public String getContent();
public void setContent(String content)
                          throws DOMException;

public String getCounterIncrement();
public void setCounterIncrement(String counterIncrement)
                          throws DOMException;

public String getCounterReset();
public void setCounterReset(String counterReset)
                          throws DOMException;

public String getCue();
public void setCue(String cue)
                          throws DOMException;

public String getCueAfter();
public void setCueAfter(String cueAfter)
                          throws DOMException;

public String getCueBefore();
public void setCueBefore(String cueBefore)
                          throws DOMException;

public String getCursor();
public void setCursor(String cursor)
                          throws DOMException;

public String getDirection();
public void setDirection(String direction)
                          throws DOMException;

public String getDisplay();
public void setDisplay(String display)
                          throws DOMException;

public String getElevation();
public void setElevation(String elevation)
                          throws DOMException;

public String getEmptyCells();
public void setEmptyCells(String emptyCells)
                          throws DOMException;

public String getCssFloat();
public void setCssFloat(String cssFloat)
                          throws DOMException;

public String getFont();
public void setFont(String font)
                          throws DOMException;
```

```
public String getFontFamily();
public void setFontFamily(String fontFamily)
                          throws DOMException;

public String getFontSize();
public void setFontSize(String fontSize)
                          throws DOMException;

public String getFontSizeAdjust();
public void setFontSizeAdjust(String fontSizeAdjust)
                          throws DOMException;

public String getFontStretch();
public void setFontStretch(String fontStretch)
                          throws DOMException;

public String getFontStyle();
public void setFontStyle(String fontStyle)
                          throws DOMException;

public String getFontVariant();
public void setFontVariant(String fontVariant)
                          throws DOMException;

public String getFontWeight();
public void setFontWeight(String fontWeight)
                          throws DOMException;

public String getHeight();
public void setHeight(String height)
                          throws DOMException;

public String getLeft();
public void setLeft(String left)
                          throws DOMException;

public String getLetterSpacing();
public void setLetterSpacing(String letterSpacing)
                          throws DOMException;

public String getLineHeight();
public void setLineHeight(String lineHeight)
                          throws DOMException;

public String getListStyle();
public void setListStyle(String listStyle)
                          throws DOMException;

public String getListStyleImage();
public void setListStyleImage(String listStyleImage)
                          throws DOMException;

public String getListStylePosition();
public void setListStylePosition(String listStylePosition)
                          throws DOMException;

public String getListStyleType();
```

```
    public void setListStyleType(String listStyleType)
                          throws DOMException;

    public String getMargin();
    public void setMargin(String margin)
                          throws DOMException;

    public String getMarginTop();
    public void setMarginTop(String marginTop)
                          throws DOMException;

    public String getMarginRight();
    public void setMarginRight(String marginRight)
                          throws DOMException;

    public String getMarginBottom();
    public void setMarginBottom(String marginBottom)
                          throws DOMException;

    public String getMarginLeft();
    public void setMarginLeft(String marginLeft)
                          throws DOMException;

    public String getMarkerOffset();
    public void setMarkerOffset(String markerOffset)
                          throws DOMException;

    public String getMarks();
    public void setMarks(String marks)
                          throws DOMException;

    public String getMaxHeight();
    public void setMaxHeight(String maxHeight)
                          throws DOMException;

    public String getMaxWidth();
    public void setMaxWidth(String maxWidth)
                          throws DOMException;

    public String getMinHeight();
    public void setMinHeight(String minHeight)
                          throws DOMException;

    public String getMinWidth();
    public void setMinWidth(String minWidth)
                          throws DOMException;

    public String getOrphans();
    public void setOrphans(String orphans)
                          throws DOMException;

    public String getOutline();
    public void setOutline(String outline)
                          throws DOMException;

    public String getOutlineColor();
    public void setOutlineColor(String outlineColor)
```

```
                          throws DOMException;

public String getOutlineStyle();
public void setOutlineStyle(String outlineStyle)
                          throws DOMException;

public String getOutlineWidth();
public void setOutlineWidth(String outlineWidth)
                          throws DOMException;

public String getOverflow();
public void setOverflow(String overflow)
                          throws DOMException;

public String getPadding();
public void setPadding(String padding)
                          throws DOMException;

public String getPaddingTop();
public void setPaddingTop(String paddingTop)
                          throws DOMException;

public String getPaddingRight();
public void setPaddingRight(String paddingRight)
                          throws DOMException;

public String getPaddingBottom();
public void setPaddingBottom(String paddingBottom)
                          throws DOMException;

public String getPaddingLeft();
public void setPaddingLeft(String paddingLeft)
                          throws DOMException;

public String getPage();
public void setPage(String page)
                          throws DOMException;

public String getPageBreakAfter();
public void setPageBreakAfter(String pageBreakAfter)
                          throws DOMException;

public String getPageBreakBefore();
public void setPageBreakBefore(String pageBreakBefore)
                          throws DOMException;

public String getPageBreakInside();
public void setPageBreakInside(String pageBreakInside)
                          throws DOMException;

public String getPause();
public void setPause(String pause)
                          throws DOMException;

public String getPauseAfter();
public void setPauseAfter(String pauseAfter)
                          throws DOMException;
```

```
    public String getPauseBefore();
    public void setPauseBefore(String pauseBefore)
                                  throws DOMException;

    public String getPitch();
    public void setPitch(String pitch)
                                  throws DOMException;

    public String getPitchRange();
    public void setPitchRange(String pitchRange)
                                  throws DOMException;

    public String getPlayDuring();
    public void setPlayDuring(String playDuring)
                                  throws DOMException;

    public String getPosition();
    public void setPosition(String position)
                                  throws DOMException;

    public String getQuotes();
    public void setQuotes(String quotes)
                                  throws DOMException;

    public String getRichness();
    public void setRichness(String richness)
                                  throws DOMException;

    public String getRight();
    public void setRight(String right)
                                  throws DOMException;

    public String getSize();
    public void setSize(String size)
                                  throws DOMException;

    public String getSpeak();
    public void setSpeak(String speak)
                                  throws DOMException;

    public String getSpeakHeader();
    public void setSpeakHeader(String speakHeader)
                                  throws DOMException;

    public String getSpeakNumeral();
    public void setSpeakNumeral(String speakNumeral)
                                  throws DOMException;

    public String getSpeakPunctuation();
    public void setSpeakPunctuation(String speakPunctuation)
                                  throws DOMException;

    public String getSpeechRate();
    public void setSpeechRate(String speechRate)
                                  throws DOMException;
```

```
public String getStress();
public void setStress(String stress)
                          throws DOMException;

public String getTableLayout();
public void setTableLayout(String tableLayout)
                          throws DOMException;

public String getTextAlign();
public void setTextAlign(String textAlign)
                          throws DOMException;

public String getTextDecoration();
public void setTextDecoration(String textDecoration)
                          throws DOMException;

public String getTextIndent();
public void setTextIndent(String textIndent)
                          throws DOMException;

public String getTextShadow();
public void setTextShadow(String textShadow)
                          throws DOMException;

public String getTextTransform();
public void setTextTransform(String textTransform)
                          throws DOMException;

public String getTop();
public void setTop(String top)
                          throws DOMException;

public String getUnicodeBidi();
public void setUnicodeBidi(String unicodeBidi)
                          throws DOMException;

public String getVerticalAlign();
public void setVerticalAlign(String verticalAlign)
                          throws DOMException;

public String getVisibility();
public void setVisibility(String visibility)
                          throws DOMException;

public String getVoiceFamily();
public void setVoiceFamily(String voiceFamily)
                          throws DOMException;

public String getVolume();
public void setVolume(String volume)
                          throws DOMException;

public String getWhiteSpace();
public void setWhiteSpace(String whiteSpace)
                          throws DOMException;

public String getWidows();
```

```
    public void setWidows(String widows)
                              throws DOMException;

    public String getWidth();
    public void setWidth(String width)
                              throws DOMException;

    public String getWordSpacing();
    public void setWordSpacing(String wordSpacing)
                              throws DOMException;

    public String getZIndex();
    public void setZIndex(String zIndex)
                              throws DOMException;

}
```

D.6: Document Object Model Events

org/w3c/dom/events/EventException.java:

```
package org.w3c.dom.events;

public class EventException extends RuntimeException {
    public EventException(short code, String message) {
       super(message);
       this.code = code;
    }
    public short   code;
    // EventExceptionCode
    public static final short UNSPECIFIED_EVENT_TYPE_ERR = 0;

}
```

org/w3c/dom/events/EventTarget.java:

```
package org.w3c.dom.events;

public interface EventTarget {
    public void addEventListener(String type,
                                 EventListener listener,
                                 boolean useCapture);

    public void removeEventListener(String type,
                                    EventListener listener,
                                    boolean useCapture);

    public boolean dispatchEvent(Event evt)
                                 throws EventException;

}
```

org/w3c/dom/events/EventListener.java:

```
package org.w3c.dom.events;

public interface EventListener {
    public void handleEvent(Event evt);

}
```

org/w3c/dom/events/Event.java:

```
package org.w3c.dom.events;

public interface Event {
    // PhaseType
    public static final short CAPTURING_PHASE           = 1;
    public static final short AT_TARGET                 = 2;
    public static final short BUBBLING_PHASE            = 3;

    public String getType();

    public EventTarget getTarget();

    public EventTarget getCurrentTarget();

    public short getEventPhase();

    public boolean getBubbles();

    public boolean getCancelable();

    public long getTimeStamp();

    public void stopPropagation();

    public void preventDefault();

    public void initEvent(String eventTypeArg,
                          boolean canBubbleArg,
                          boolean cancelableArg);

}
```

org/w3c/dom/events/DocumentEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.DOMException;

public interface DocumentEvent {
    public Event createEvent(String eventType)
                             throws DOMException;

}
```

org/w3c/dom/events/UIEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.views.AbstractView;

public interface UIEvent extends Event {
    public AbstractView getView();

    public int getDetail();

    public void initUIEvent(String typeArg,
                            boolean canBubbleArg,
                            boolean cancelableArg,
                            AbstractView viewArg,
                            int detailArg);

}
```

org/w3c/dom/events/MouseEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.views.AbstractView;

public interface MouseEvent extends UIEvent {
    public int getScreenX();

    public int getScreenY();

    public int getClientX();

    public int getClientY();

    public boolean getCtrlKey();

    public boolean getShiftKey();

    public boolean getAltKey();

    public boolean getMetaKey();

    public short getButton();

    public EventTarget getRelatedTarget();

    public void initMouseEvent(String typeArg,
                               boolean canBubbleArg,
                               boolean cancelableArg,
                               AbstractView viewArg,
                               int detailArg,
                               int screenXArg,
                               int screenYArg,
                               int clientXArg,
                               int clientYArg,
                               boolean ctrlKeyArg,
                               boolean altKeyArg,
                               boolean shiftKeyArg,
                               boolean metaKeyArg,
                               short buttonArg,
                               EventTarget relatedTargetArg);

}
```

org/w3c/dom/events/MutationEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.Node;

public interface MutationEvent extends Event {
    public Node getRelatedNode();

    public String getPrevValue();

    public String getNewValue();

    public String getAttrName();

    public void initMutationEvent(String typeArg,
                                  boolean canBubbleArg,
                                  boolean cancelableArg,
                                  Node relatedNodeArg,
                                  String prevValueArg,
                                  String newValueArg,
                                  String attrNameArg);

}
```

D.7: Document Object Model Traversal

org/w3c/dom/traversal/NodeIterator.java:

```
package org.w3c.dom.traversal;

import org.w3c.dom.DOMException;
import org.w3c.dom.Node;

public interface NodeIterator {
    public Node getRoot();

    public int getWhatToShow();

    public NodeFilter getFilter();

    public boolean getExpandEntityReferences();

    public Node nextNode()
                        throws DOMException;

    public Node previousNode()
```

```
                             throws DOMException;

    public void detach();

}
```

org/w3c/dom/traversal/NodeFilter.java:

```
package org.w3c.dom.traversal;

import org.w3c.dom.Node;

public interface NodeFilter {
    // Constants returned by acceptNode
    public static final short FILTER_ACCEPT             = 1;
    public static final short FILTER_REJECT             = 2;
    public static final short FILTER_SKIP               = 3;

    // Constants for whatToShow
    public static final int SHOW_ALL                  = 0xFFFFFFFF;
    public static final int SHOW_ELEMENT              = 0x00000001;
    public static final int SHOW_ATTRIBUTE            = 0x00000002;
    public static final int SHOW_TEXT                 = 0x00000004;
    public static final int SHOW_CDATA_SECTION        = 0x00000008;
    public static final int SHOW_ENTITY_REFERENCE     = 0x00000010;
    public static final int SHOW_ENTITY               = 0x00000020;
    public static final int SHOW_PROCESSING_INSTRUCTION = 0x00000040;
    public static final int SHOW_COMMENT              = 0x00000080;
    public static final int SHOW_DOCUMENT             = 0x00000100;
    public static final int SHOW_DOCUMENT_TYPE        = 0x00000200;
    public static final int SHOW_DOCUMENT_FRAGMENT    = 0x00000400;
    public static final int SHOW_NOTATION             = 0x00000800;

    public short acceptNode(Node n);

}
```

org/w3c/dom/traversal/TreeWalker.java:

```
package org.w3c.dom.traversal;

import org.w3c.dom.DOMException;
import org.w3c.dom.Node;

public interface TreeWalker {
    public Node getRoot();

    public int getWhatToShow();

    public NodeFilter getFilter();

    public boolean getExpandEntityReferences();

    public Node getCurrentNode();
    public void setCurrentNode(Node currentNode)
                            throws DOMException;
    public Node parentNode();

    public Node firstChild();

    public Node lastChild();

    public Node previousSibling();

    public Node nextSibling();

    public Node previousNode();

    public Node nextNode();

}
```

org/w3c/dom/traversal/DocumentTraversal.java:

```
package org.w3c.dom.traversal;

import org.w3c.dom.DOMException;
import org.w3c.dom.Node;

public interface DocumentTraversal {
    public NodeIterator createNodeIterator(Node root,
                                           int whatToShow,
                                           NodeFilter filter,
                                           boolean entityReferenceExpansion)
                                           throws DOMException;

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

}
```

D.8: Document Object Model Range

org/w3c/dom/range/RangeException.java:

```
package org.w3c.dom.range;

public class RangeException extends RuntimeException {
    public RangeException(short code, String message) {
       super(message);
       this.code = code;
    }
    public short   code;
    // RangeExceptionCode
    public static final short BAD_BOUNDARYPOINTS_ERR    = 1;
    public static final short INVALID_NODE_TYPE_ERR     = 2;

}
```

org/w3c/dom/range/Range.java:

```
package org.w3c.dom.range;

import org.w3c.dom.DOMException;
import org.w3c.dom.DocumentFragment;
import org.w3c.dom.Node;

public interface Range {
    public Node getStartContainer()
                                       throws DOMException;

    public int getStartOffset()
                                       throws DOMException;

    public Node getEndContainer()
                                       throws DOMException;

    public int getEndOffset()
                                       throws DOMException;

    public boolean getCollapsed()
                                       throws DOMException;

    public Node getCommonAncestorContainer()
                                       throws DOMException;

    public void setStart(Node refNode,
                         int offset)
                         throws RangeException, DOMException;

    public void setEnd(Node refNode,
                       int offset)
                       throws RangeException, DOMException;

    public void setStartBefore(Node refNode)
                               throws RangeException, DOMException;

    public void setStartAfter(Node refNode)
                              throws RangeException, DOMException;

    public void setEndBefore(Node refNode)
                             throws RangeException, DOMException;

    public void setEndAfter(Node refNode)
                            throws RangeException, DOMException;

    public void collapse(boolean toStart)
                         throws DOMException;

    public void selectNode(Node refNode)
```

```
                       throws RangeException, DOMException;

    public void selectNodeContents(Node refNode)
                               throws RangeException, DOMException;

    // CompareHow
    public static final short START_TO_START            = 0;
    public static final short START_TO_END              = 1;
    public static final short END_TO_END                = 2;
    public static final short END_TO_START              = 3;

    public short compareBoundaryPoints(short how,
                                       Range sourceRange)
                                       throws DOMException;

    public void deleteContents()
                           throws DOMException;

    public DocumentFragment extractContents()
                                        throws DOMException;

    public DocumentFragment cloneContents()
                                      throws DOMException;

    public void insertNode(Node newNode)
                       throws DOMException, RangeException;

    public void surroundContents(Node newParent)
                             throws DOMException, RangeException;

    public Range cloneRange()
                        throws DOMException;

    public String toString()
                       throws DOMException;

    public void detach()
                   throws DOMException;

}
```

org/w3c/dom/range/DocumentRange.java:

```
package org.w3c.dom.range;

public interface DocumentRange {
    public Range createRange();

}
```

org/w3c/dom/range/DocumentRange.java:

Appendix E: ECMA Script Language Binding

This appendix contains the complete ECMA Script binding for the Level 2 Document Object Model definitions. The definitions are divided into Core [p.395] , HTML [p.401] , StyleSheets [p.422] , CSS [p.423] , Events [p.438] , Traversal [p.441] , and Range [p.442] .

E.1: Document Object Model Core

Object **DOMString**
Object **DOMTimeStamp**
Object **DOMImplementation**
- The **DOMImplementation** object has the following methods:
  - **hasFeature(feature, version)**
    - This method returns a **boolean**. The **feature** parameter is of type **DOMString**. The **version** parameter is of type **DOMString**.
  - **createDocumentType(qualifiedName, publicId, systemId)**
    - This method returns a **DocumentType**. The **qualifiedName** parameter is of type **DOMString**. The **publicId** parameter is of type **DOMString**. The **systemId** parameter is of type **DOMString**.
  - **createDocument(namespaceURI, qualifiedName, doctype)**
    - This method returns a **Document**. The **namespaceURI** parameter is of type **DOMString**. The **qualifiedName** parameter is of type **DOMString**. The **doctype** parameter is of type **DocumentType**.

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

Object **Document**
- **Document** has the all the properties and methods of **Node** as well as the properties and methods defined below.
- The **Document** object has the following properties:
  - **doctype**
    - This property is of type **DocumentType**.
  - **implementation**
    - This property is of type **DOMImplementation**.
  - **documentElement**
    - This property is of type **Element**.
- The **Document** object has the following methods:
  - **createElement(tagName)**
    - This method returns a **Element**. The **tagName** parameter is of type **DOMString**.
  - **createDocumentFragment()**
    - This method returns a **DocumentFragment**.
  - **createTextNode(data)**
    - This method returns a **Text**. The **data** parameter is of type **DOMString**.
  - **createComment(data)**
    - This method returns a **Comment**. The **data** parameter is of type **DOMString**.

createCDATASection(data)
This method returns a **CDATASection**. The **data** parameter is of type **DOMString**.

createProcessingInstruction(target, data)
This method returns a **ProcessingInstruction**. The **target** parameter is of type **DOMString**. The **data** parameter is of type **DOMString**.

createAttribute(name)
This method returns a **Attr**. The **name** parameter is of type **DOMString**.

createEntityReference(name)
This method returns a **EntityReference**. The **name** parameter is of type **DOMString**.

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

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

createElementNS(namespaceURI, qualifiedName)
This method returns a **Element**. The **namespaceURI** parameter is of type **DOMString**. The **qualifiedName** parameter is of type **DOMString**.

createAttributeNS(namespaceURI, qualifiedName)
This method returns a **Attr**. The **namespaceURI** parameter is of type **DOMString**. The **qualifiedName** parameter is of type **DOMString**.

getElementsByTagNameNS(namespaceURI, localName)
This method returns a **NodeList**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.

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

Class **Node**

The **Node** class has the following constants:

Node.ELEMENT_NODE
This constant is of type **short** and its value is **1**.

Node.ATTRIBUTE_NODE
This constant is of type **short** and its value is **2**.

Node.TEXT_NODE
This constant is of type **short** and its value is **3**.

Node.CDATA_SECTION_NODE
This constant is of type **short** and its value is **4**.

Node.ENTITY_REFERENCE_NODE
This constant is of type **short** and its value is **5**.

Node.ENTITY_NODE
This constant is of type **short** and its value is **6**.

Node.PROCESSING_INSTRUCTION_NODE
This constant is of type **short** and its value is **7**.

Node.COMMENT_NODE
This constant is of type **short** and its value is **8**.

Node.DOCUMENT_NODE
This constant is of type **short** and its value is **9**.

Node.DOCUMENT_TYPE_NODE
This constant is of type **short** and its value is **10**.
Node.DOCUMENT_FRAGMENT_NODE
This constant is of type **short** and its value is **11**.
Node.NOTATION_NODE
This constant is of type **short** and its value is **12**.

Object **Node**

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

nodeName
This property is of type **String**.
nodeValue
This property is of type **String**.
nodeType
This property is of type **short**.
parentNode
This property is of type **Node**.
childNodes
This property is of type **NodeList**.
firstChild
This property is of type **Node**.
lastChild
This property is of type **Node**.
previousSibling
This property is of type **Node**.
nextSibling
This property is of type **Node**.
attributes
This property is of type **NamedNodeMap**.
ownerDocument
This property is of type **Document**.
namespaceURI
This property is of type **String**.
prefix
This property is of type **String**.
localName
This property is of type **String**.

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

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

appendChild(newChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**.
hasChildNodes()
This method returns a **boolean**.
cloneNode(deep)
This method returns a **Node**. The **deep** parameter is of type **boolean**.
normalize()
This method returns a **void**.
supports(feature, version)
This method returns a **boolean**. The **feature** parameter is of type **DOMString**. The **version** parameter is of type **DOMString**.

Object **NodeList**
The **NodeList** object has the following properties:
length
This property is of type **int**.
The **NodeList** object has the following methods:
item(index)
This method returns a **Node**. The **index** parameter is of type **unsigned long**. 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 property is of type **int**.
The **NamedNodeMap** object has the following methods:
getNamedItem(name)
This method returns a **Node**. The **name** parameter is of type **DOMString**.
setNamedItem(arg)
This method returns a **Node**. The **arg** parameter is of type **Node**.
removeNamedItem(name)
This method returns a **Node**. The **name** parameter is of type **DOMString**.
item(index)
This method returns a **Node**. The **index** parameter is of type **unsigned long**. 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.
getNamedItemNS(namespaceURI, localName)
This method returns a **Node**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.
setNamedItemNS(arg)
This method returns a **Node**. The **arg** parameter is of type **Node**.
removeNamedItemNS(namespaceURI, localName)
This method returns a **Node**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.

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

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

data

This property is of type **String**.

length

This property is of type **int**.

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

substringData(offset, count)

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

appendData(arg)

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

insertData(offset, arg)

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

deleteData(offset, count)

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

replaceData(offset, count, arg)

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

Object **Attr**

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

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

name

This property is of type **String**.

specified

This property is of type **boolean**.

value

This property is of type **String**.

ownerElement

This property is of type **Element**.

Object **Element**

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

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

tagName

This property is of type **String**.

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

getAttribute(name)

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

setAttribute(name, value)

This method returns a **void**. The **name** parameter is of type **DOMString**. The **value** parameter is of type **DOMString**.

removeAttribute(name)

This method returns a **void**. The **name** parameter is of type **DOMString**.

getAttributeNode(name)
This method returns a **Attr**. The **name** parameter is of type **DOMString**.
setAttributeNode(newAttr)
This method returns a **Attr**. The **newAttr** parameter is of type **Attr**.
removeAttributeNode(oldAttr)
This method returns a **Attr**. The **oldAttr** parameter is of type **Attr**.
getElementsByTagName(name)
This method returns a **NodeList**. The **name** parameter is of type **DOMString**.
getAttributeNS(namespaceURI, localName)
This method returns a **DOMString**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.
setAttributeNS(namespaceURI, qualifiedName, value)
This method returns a **void**. The **namespaceURI** parameter is of type **DOMString**. The **qualifiedName** parameter is of type **DOMString**. The **value** parameter is of type **DOMString**.
removeAttributeNS(namespaceURI, localName)
This method returns a **void**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.
getAttributeNodeNS(namespaceURI, localName)
This method returns a **Attr**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.
setAttributeNodeNS(newAttr)
This method returns a **Attr**. The **newAttr** parameter is of type **Attr**.
getElementsByTagNameNS(namespaceURI, localName)
This method returns a **NodeList**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.
hasAttribute(name)
This method returns a **boolean**. The **name** parameter is of type **DOMString**.
hasAttributeNS(namespaceURI, localName)
This method returns a **boolean**. The **namespaceURI** parameter is of type **DOMString**. The **localName** parameter is of type **DOMString**.

Object **Text**
Text has the all the properties and methods of **CharacterData** as well as the properties and methods defined below.
The **Text** object has the following methods:
splitText(offset)
This method returns a **Text**. The **offset** parameter is of type **unsigned long**.

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

Object **CDATASection**
CDATASection has the all the properties and methods of **Text** as well as the properties and methods defined below.

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

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

name

This property is of type **String**.

entities

This property is of type **NamedNodeMap**.

notations

This property is of type **NamedNodeMap**.

publicId

This property is of type **String**.

systemId

This property is of type **String**.

internalSubset

This property is of type **String**.

Object **Notation**

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

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

publicId

This property is of type **String**.

systemId

This property is of type **String**.

Object **Entity**

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

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

publicId

This property is of type **String**.

systemId

This property is of type **String**.

notationName

This property is of type **String**.

Object **EntityReference**

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

Object **ProcessingInstruction**

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

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

target

This property is of type **String**.

data

This property is of type **String**.

E.2: Document Object Model HTML

Object **HTMLDOMImplementation**

HTMLDOMImplementation has the all the properties and methods of **DOMImplementation** as well as the properties and methods defined below.

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

createHTMLDocument(title)

This method returns a **HTMLDocument**. The **title** parameter is of type **DOMString**.

Object **HTMLCollection**

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

length

This property is of type **int**.

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

item(index)

This method returns a **Node**. The **index** parameter is of type **unsigned long**. 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**. The **name** parameter is of type **DOMString**. 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 **Document** as well as the properties and methods defined below.

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

title

This property is of type **String**.

referrer

This property is of type **String**.

domain

This property is of type **String**.

URL

This property is of type **String**.

body

This property is of type **HTMLElement**.

images

This property is of type **HTMLCollection**.

applets

This property is of type **HTMLCollection**.

links

This property is of type **HTMLCollection**.

forms

This property is of type **HTMLCollection**.

anchors

This property is of type **HTMLCollection**.

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

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

open()
This method returns a **void**.

close()
This method returns a **void**.

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

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

getElementsByName(elementName)
This method returns a **NodeList**. The **elementName** parameter is of type **DOMString**.

Object **HTMLElement**

HTMLElement has the all the properties and methods of **Element** as well as the properties and methods defined below.

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

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

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

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

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

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

Object **HTMLHtmlElement**

HTMLHtmlElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLHeadElement**

HTMLHeadElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLLinkElement**

HTMLLinkElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

charset
This property is of type **String**.
href
This property is of type **String**.
hreflang
This property is of type **String**.
media
This property is of type **String**.
rel
This property is of type **String**.
rev
This property is of type **String**.
target
This property is of type **String**.
type
This property is of type **String**.

Object **HTMLTitleElement**

HTMLTitleElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLMetaElement**

HTMLMetaElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

content
This property is of type **String**.
httpEquiv
This property is of type **String**.
name
This property is of type **String**.
scheme
This property is of type **String**.

Object **HTMLBaseElement**

HTMLBaseElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLIsIndexElement**

HTMLIsIndexElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

form

This property is of type **HTMLFormElement**.

prompt

This property is of type **String**.

Object **HTMLStyleElement**

HTMLStyleElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

disabled

This property is of type **boolean**.

media

This property is of type **String**.

type

This property is of type **String**.

Object **HTMLBodyElement**

HTMLBodyElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

aLink

This property is of type **String**.

background

This property is of type **String**.

bgColor

This property is of type **String**.

link

This property is of type **String**.

text

This property is of type **String**.

vLink

This property is of type **String**.

Object **HTMLFormElement**

HTMLFormElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

elements

This property is of type **HTMLCollection**.

length

This property is of type **long**.

name

This property is of type **String**.

acceptCharset

This property is of type **String**.

action

This property is of type **String**.

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

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

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

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

submit()
: This method returns a **void**.

reset()
: This method returns a **void**.

Object **HTMLSelectElement**

HTMLSelectElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

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

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

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

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

options
: This property is of type **HTMLCollection**.

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

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

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

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

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

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

add(element, before)
: This method returns a **void**. The **element** parameter is of type **HTMLElement**. The **before** parameter is of type **HTMLElement**.

remove(index)
: This method returns a **void**. The **index** parameter is of type **long**.

blur()
: This method returns a **void**.

focus()
This method returns a **void**.

Object **HTMLOptGroupElement**

HTMLOptGroupElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

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

Object **HTMLOptionElement**

HTMLOptionElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

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

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

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

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

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

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

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

Object **HTMLInputElement**

HTMLInputElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

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

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

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

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

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

alt
This property is of type **String**.
checked
This property is of type **boolean**.
disabled
This property is of type **boolean**.
maxLength
This property is of type **long**.
name
This property is of type **String**.
readOnly
This property is of type **boolean**.
size
This property is of type **String**.
src
This property is of type **String**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
useMap
This property is of type **String**.
value
This property is of type **String**.

The **HTMLInputElement** object has the following methods:
blur()
This method returns a **void**.
focus()
This method returns a **void**.
select()
This method returns a **void**.
click()
This method returns a **void**.

Object **HTMLTextAreaElement**

HTMLTextAreaElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTextAreaElement** object has the following properties:
defaultValue
This property is of type **String**.
form
This property is of type **HTMLFormElement**.
accessKey
This property is of type **String**.
cols
This property is of type **long**.

disabled
This property is of type **boolean**.
name
This property is of type **String**.
readOnly
This property is of type **boolean**.
rows
This property is of type **long**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
value
This property is of type **String**.

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

blur()
This method returns a **void**.
focus()
This method returns a **void**.
select()
This method returns a **void**.

Object **HTMLButtonElement**

HTMLButtonElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

form
This property is of type **HTMLFormElement**.
accessKey
This property is of type **String**.
disabled
This property is of type **boolean**.
name
This property is of type **String**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
value
This property is of type **String**.

Object **HTMLLabelElement**

HTMLLabelElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

accessKey

This property is of type **String**.

htmlFor

This property is of type **String**.

Object **HTMLFieldSetElement**

HTMLFieldSetElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

form

This property is of type **HTMLFormElement**.

Object **HTMLLegendElement**

HTMLLegendElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

form

This property is of type **HTMLFormElement**.

accessKey

This property is of type **String**.

align

This property is of type **String**.

Object **HTMLUListElement**

HTMLUListElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

compact

This property is of type **boolean**.

type

This property is of type **String**.

Object **HTMLOListElement**

HTMLOListElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

compact

This property is of type **boolean**.

start

This property is of type **long**.

type

This property is of type **String**.

Object **HTMLDListElement**

HTMLDListElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

compact

This property is of type **boolean**.

Object **HTMLDirectoryElement**

HTMLDirectoryElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

compact

This property is of type **boolean**.

Object **HTMLMenuElement**

HTMLMenuElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

compact

This property is of type **boolean**.

Object **HTMLLIElement**

HTMLLIElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

type

This property is of type **String**.

value

This property is of type **long**.

Object **HTMLDivElement**

HTMLDivElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

align

This property is of type **String**.

Object **HTMLParagraphElement**

HTMLParagraphElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

align

This property is of type **String**.

Object **HTMLHeadingElement**

HTMLHeadingElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

align

This property is of type **String**.

Object **HTMLQuoteElement**

HTMLQuoteElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

cite

This property is of type **String**.

Object **HTMLPreElement**

HTMLPreElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

width

This property is of type **long**.

Object **HTMLBRElement**

HTMLBRElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

clear

This property is of type **String**.

Object **HTMLBaseFontElement**

HTMLBaseFontElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

color

This property is of type **String**.

face

This property is of type **String**.

size

This property is of type **String**.

Object **HTMLFontElement**

HTMLFontElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

color

This property is of type **String**.

face

This property is of type **String**.

size

This property is of type **String**.

Object **HTMLHRElement**

HTMLHRElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

align

This property is of type **String**.

noShade

This property is of type **boolean**.

size

This property is of type **String**.

width

This property is of type **String**.

Object **HTMLModElement**

HTMLModElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

cite

This property is of type **String**.

dateTime

This property is of type **String**.

Object **HTMLAnchorElement**

HTMLAnchorElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

accessKey

This property is of type **String**.

charset

This property is of type **String**.

coords

This property is of type **String**.

href

This property is of type **String**.

hreflang

This property is of type **String**.

name

This property is of type **String**.

rel

This property is of type **String**.

rev

This property is of type **String**.

shape

This property is of type **String**.

tabIndex

This property is of type **long**.

target

This property is of type **String**.

type

This property is of type **String**.

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

blur()

This method returns a **void**.

focus()

This method returns a **void**.

Object **HTMLImageElement**

HTMLImageElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

lowSrc

This property is of type **String**.

name

This property is of type **String**.

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

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

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

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

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

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

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

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

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

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

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

Object **HTMLObjectElement**

HTMLObjectElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

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

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

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

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

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

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

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

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

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

hspace
This property is of type **String**.
name
This property is of type **String**.
standby
This property is of type **String**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
useMap
This property is of type **String**.
vspace
This property is of type **String**.
width
This property is of type **String**.
contentDocument
This property is of type **Document**.

Object **HTMLParamElement**

HTMLParamElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLAppletElement**

HTMLAppletElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

align
This property is of type **String**.
alt
This property is of type **String**.
archive
This property is of type **String**.
code
This property is of type **String**.
codeBase
This property is of type **String**.
height
This property is of type **String**.

hspace
This property is of type **String**.
name
This property is of type **String**.
object
This property is of type **String**.
vspace
This property is of type **String**.
width
This property is of type **String**.

Object **HTMLMapElement**

HTMLMapElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLAreaElement**

HTMLAreaElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

accessKey
This property is of type **String**.
alt
This property is of type **String**.
coords
This property is of type **String**.
href
This property is of type **String**.
noHref
This property is of type **boolean**.
shape
This property is of type **String**.
tabIndex
This property is of type **long**.
target
This property is of type **String**.

Object **HTMLScriptElement**

HTMLScriptElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

event
This property is of type **String**.
charset
This property is of type **String**.
defer
This property is of type **boolean**.
src
This property is of type **String**.
type
This property is of type **String**.

Object **HTMLTableElement**

HTMLTableElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

caption
This property is of type **HTMLTableCaptionElement**.
tHead
This property is of type **HTMLTableSectionElement**.
tFoot
This property is of type **HTMLTableSectionElement**.
rows
This property is of type **HTMLCollection**.
tBodies
This property is of type **HTMLCollection**.
align
This property is of type **String**.
bgColor
This property is of type **String**.
border
This property is of type **String**.
cellPadding
This property is of type **String**.
cellSpacing
This property is of type **String**.
frame
This property is of type **String**.
rules
This property is of type **String**.
summary
This property is of type **String**.
width
This property is of type **String**.

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

createTHead()
This method returns a **HTMLElement**.

deleteTHead()
This method returns a **void**.
createTFoot()
This method returns a **HTMLElement**.
deleteTFoot()
This method returns a **void**.
createCaption()
This method returns a **HTMLElement**.
deleteCaption()
This method returns a **void**.
insertRow(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteRow(index)
This method returns a **void**. The **index** parameter is of type **long**.

Object **HTMLTableCaptionElement**
HTMLTableCaptionElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLTableCaptionElement** object has the following properties:
align
This property is of type **String**.

Object **HTMLTableColElement**
HTMLTableColElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLTableColElement** object has the following properties:
align
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
span
This property is of type **long**.
vAlign
This property is of type **String**.
width
This property is of type **String**.

Object **HTMLTableSectionElement**
HTMLTableSectionElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLTableSectionElement** object has the following properties:
align
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.

vAlign
This property is of type **String**.
rows
This property is of type **HTMLCollection**.
The **HTMLTableSectionElement** object has the following methods:
insertRow(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteRow(index)
This method returns a **void**. The **index** parameter is of type **long**.
Object **HTMLTableRowElement**
HTMLTableRowElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLTableRowElement** object has the following properties:
rowIndex
This property is of type **long**.
sectionRowIndex
This property is of type **long**.
cells
This property is of type **HTMLCollection**.
align
This property is of type **String**.
bgColor
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
vAlign
This property is of type **String**.
The **HTMLTableRowElement** object has the following methods:
insertCell(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteCell(index)
This method returns a **void**. The **index** parameter is of type **long**.
Object **HTMLTableCellElement**
HTMLTableCellElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLTableCellElement** object has the following properties:
cellIndex
This property is of type **long**.
abbr
This property is of type **String**.
align
This property is of type **String**.
axis
This property is of type **String**.

bgColor
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
colSpan
This property is of type **long**.
headers
This property is of type **String**.
height
This property is of type **String**.
noWrap
This property is of type **boolean**.
rowSpan
This property is of type **long**.
scope
This property is of type **String**.
vAlign
This property is of type **String**.
width
This property is of type **String**.

Object **HTMLFrameSetElement**

HTMLFrameSetElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

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

Object **HTMLFrameElement**

HTMLFrameElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

frameBorder
This property is of type **String**.
longDesc
This property is of type **String**.
marginHeight
This property is of type **String**.
marginWidth
This property is of type **String**.
name
This property is of type **String**.
noResize
This property is of type **boolean**.

scrolling
This property is of type **String**.
src
This property is of type **String**.
contentDocument
This property is of type **Document**.

Object **HTMLIFrameElement**

HTMLIFrameElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

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

align
This property is of type **String**.
frameBorder
This property is of type **String**.
height
This property is of type **String**.
longDesc
This property is of type **String**.
marginHeight
This property is of type **String**.
marginWidth
This property is of type **String**.
name
This property is of type **String**.
scrolling
This property is of type **String**.
src
This property is of type **String**.
width
This property is of type **String**.
contentDocument
This property is of type **Document**.

E.3: Document Object Model Views

Object **AbstractView**

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

document
This property is of type **DocumentView**.

Object **DocumentView**

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

defaultView
This property is of type **AbstractView**.

E.4: Document Object Model StyleSheets

Object **StyleSheet**

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

type

This property is of type **String**.

disabled

This property is of type **boolean**.

ownerNode

This property is of type **Node**.

parentStyleSheet

This property is of type **StyleSheet**.

href

This property is of type **String**.

title

This property is of type **String**.

media

This property is of type **MediaList**.

Object **StyleSheetList**

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

length

This property is of type **int**.

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

item(index)

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

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

mediaText

This property is of type **String**.

length

This property is of type **int**.

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

item(index)

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

deleteMedium(oldMedium)

This method returns a **void**. The **oldMedium** parameter is of type **DOMString**.

appendMedium(newMedium)

This method returns a **void**. The **newMedium** parameter is of type **DOMString**.

Object **LinkStyle**

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

sheet
This property is of type **StyleSheet**.

Object **DocumentStyle**
The **DocumentStyle** object has the following properties:
styleSheets
This property is of type **StyleSheetList**.

E.5: Document Object Model CSS

Object **CSSStyleSheet**
CSSStyleSheet has the all the properties and methods of **StyleSheet** as well as the properties and methods defined below.
The **CSSStyleSheet** object has the following properties:
ownerRule
This property is of type **CSSRule**.
cssRules
This property is of type **CSSRuleList**.
The **CSSStyleSheet** object has the following methods:
insertRule(rule, index)
This method returns a **unsigned long**. The **rule** parameter is of type **DOMString**. The **index** parameter is of type **unsigned long**.
deleteRule(index)
This method returns a **void**. The **index** parameter is of type **unsigned long**.

Object **CSSRuleList**
The **CSSRuleList** object has the following properties:
length
This property is of type **int**.
The **CSSRuleList** object has the following methods:
item(index)
This method returns a **CSSRule**. The **index** parameter is of type **unsigned long**. 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.

Class **CSSRule**
The **CSSRule** class has the following constants:
CSSRule.UNKNOWN_RULE
This constant is of type **short** and its value is **0**.
CSSRule.STYLE_RULE
This constant is of type **short** and its value is **1**.
CSSRule.CHARSET_RULE
This constant is of type **short** and its value is **2**.
CSSRule.IMPORT_RULE
This constant is of type **short** and its value is **3**.
CSSRule.MEDIA_RULE
This constant is of type **short** and its value is **4**.

CSSRule.FONT_FACE_RULE
This constant is of type **short** and its value is **5**.
CSSRule.PAGE_RULE
This constant is of type **short** and its value is **6**.

Object **CSSRule**
The **CSSRule** object has the following properties:
type
This property is of type **short**.
cssText
This property is of type **String**.
parentStyleSheet
This property is of type **CSSStyleSheet**.
parentRule
This property is of type **CSSRule**.

Object **CSSStyleRule**
CSSStyleRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.
The **CSSStyleRule** object has the following properties:
selectorText
This property is of type **String**.
style
This property is of type **CSSStyleDeclaration**.

Object **CSSMediaRule**
CSSMediaRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.
The **CSSMediaRule** object has the following properties:
media
This property is of type **MediaList**.
cssRules
This property is of type **CSSRuleList**.
The **CSSMediaRule** object has the following methods:
insertRule(rule, index)
This method returns a **unsigned long**. The **rule** parameter is of type **DOMString**. The **index** parameter is of type **unsigned long**.
deleteRule(index)
This method returns a **void**. The **index** parameter is of type **unsigned long**.

Object **CSSFontFaceRule**
CSSFontFaceRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.
The **CSSFontFaceRule** object has the following properties:
style
This property is of type **CSSStyleDeclaration**.

Object **CSSPageRule**
CSSPageRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.

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

selectorText

This property is of type **String**.

style

This property is of type **CSSStyleDeclaration**.

Object **CSSImportRule**

CSSImportRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.

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

href

This property is of type **String**.

media

This property is of type **MediaList**.

styleSheet

This property is of type **CSSStyleSheet**.

Object **CSSCharsetRule**

CSSCharsetRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.

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

encoding

This property is of type **String**.

Object **CSSUnknownRule**

CSSUnknownRule has the all the properties and methods of **CSSRule** as well as the properties and methods defined below.

Object **CSSStyleDeclaration**

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

cssText

This property is of type **String**.

length

This property is of type **int**.

parentRule

This property is of type **CSSRule**.

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

getPropertyValue(propertyName)

This method returns a **DOMString**. The **propertyName** parameter is of type **DOMString**.

getPropertyCSSValue(propertyName)

This method returns a **CSSValue**. The **propertyName** parameter is of type **DOMString**.

removeProperty(propertyName)

This method returns a **DOMString**. The **propertyName** parameter is of type **DOMString**.

getPropertyPriority(propertyName)

This method returns a **DOMString**. The **propertyName** parameter is of type **DOMString**.

setProperty(propertyName, value, priority)

This method returns a **void**. The **propertyName** parameter is of type **DOMString**. The **value** parameter is of type **DOMString**. The **priority** parameter is of type **DOMString**.

item(index)

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

Class **CSSValue**

The **CSSValue** class has the following constants:

CSSValue.CSS_INHERIT
This constant is of type **short** and its value is **0**.

CSSValue.CSS_PRIMITIVE_VALUE
This constant is of type **short** and its value is **1**.

CSSValue.CSS_VALUE_LIST
This constant is of type **short** and its value is **2**.

CSSValue.CSS_CUSTOM
This constant is of type **short** and its value is **3**.

Object **CSSValue**

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

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

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

Class **CSSPrimitiveValue**

The **CSSPrimitiveValue** class has the following constants:

CSSPrimitiveValue.CSS_UNKNOWN
This constant is of type **short** and its value is **0**.

CSSPrimitiveValue.CSS_NUMBER
This constant is of type **short** and its value is **1**.

CSSPrimitiveValue.CSS_PERCENTAGE
This constant is of type **short** and its value is **2**.

CSSPrimitiveValue.CSS_EMS
This constant is of type **short** and its value is **3**.

CSSPrimitiveValue.CSS_EXS
This constant is of type **short** and its value is **4**.

CSSPrimitiveValue.CSS_PX
This constant is of type **short** and its value is **5**.

CSSPrimitiveValue.CSS_CM
This constant is of type **short** and its value is **6**.

CSSPrimitiveValue.CSS_MM
This constant is of type **short** and its value is **7**.

CSSPrimitiveValue.CSS_IN
This constant is of type **short** and its value is **8**.

CSSPrimitiveValue.CSS_PT
This constant is of type **short** and its value is **9**.

CSSPrimitiveValue.CSS_PC
This constant is of type **short** and its value is **10**.

CSSPrimitiveValue.CSS_DEG
This constant is of type **short** and its value is **11**.

CSSPrimitiveValue.CSS_RAD
This constant is of type **short** and its value is **12**.

CSSPrimitiveValue.CSS_GRAD
This constant is of type **short** and its value is **13**.
CSSPrimitiveValue.CSS_MS
This constant is of type **short** and its value is **14**.
CSSPrimitiveValue.CSS_S
This constant is of type **short** and its value is **15**.
CSSPrimitiveValue.CSS_HZ
This constant is of type **short** and its value is **16**.
CSSPrimitiveValue.CSS_KHZ
This constant is of type **short** and its value is **17**.
CSSPrimitiveValue.CSS_DIMENSION
This constant is of type **short** and its value is **18**.
CSSPrimitiveValue.CSS_STRING
This constant is of type **short** and its value is **19**.
CSSPrimitiveValue.CSS_URI
This constant is of type **short** and its value is **20**.
CSSPrimitiveValue.CSS_IDENT
This constant is of type **short** and its value is **21**.
CSSPrimitiveValue.CSS_ATTR
This constant is of type **short** and its value is **22**.
CSSPrimitiveValue.CSS_COUNTER
This constant is of type **short** and its value is **23**.
CSSPrimitiveValue.CSS_RECT
This constant is of type **short** and its value is **24**.
CSSPrimitiveValue.CSS_RGBCOLOR
This constant is of type **short** and its value is **25**.

Object **CSSPrimitiveValue**

CSSPrimitiveValue has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

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

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

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

setFloatValue(unitType, floatValue)
This method returns a **void**. The **unitType** parameter is of type **unsigned short**. The **floatValue** parameter is of type **float**.
getFloatValue(unitType)
This method returns a **float**. The **unitType** parameter is of type **unsigned short**.
setStringValue(stringType, stringValue)
This method returns a **void**. The **stringType** parameter is of type **unsigned short**. The **stringValue** parameter is of type **DOMString**.
getStringValue()
This method returns a **DOMString**.
getCounterValue()
This method returns a **Counter**.

getRectValue()

This method returns a **Rect**.

getRGBColorValue()

This method returns a **RGBColor**.

Object **CSSValueList**

CSSValueList has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

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

length

This property is of type **int**.

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

item(index)

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

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

red

This property is of type **CSSPrimitiveValue**.

green

This property is of type **CSSPrimitiveValue**.

blue

This property is of type **CSSPrimitiveValue**.

Object **Rect**

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

top

This property is of type **CSSPrimitiveValue**.

right

This property is of type **CSSPrimitiveValue**.

bottom

This property is of type **CSSPrimitiveValue**.

left

This property is of type **CSSPrimitiveValue**.

Object **Counter**

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

identifier

This property is of type **String**.

listStyle

This property is of type **String**.

separator

This property is of type **String**.

Object **ViewCSS**

ViewCSS has the all the properties and methods of **AbstractView** as well as the properties and methods defined below.

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

getComputedStyle(elt, pseudoElt)
This method returns a **CSSStyleDeclaration**. The **elt** parameter is of type **Element**. The **pseudoElt** parameter is of type **DOMString**.

Object **DocumentCSS**
DocumentCSS has the all the properties and methods of **DocumentStyle** as well as the properties and methods defined below.
The **DocumentCSS** object has the following methods:
getOverrideStyle(elt, pseudoElt)
This method returns a **CSSStyleDeclaration**. The **elt** parameter is of type **Element**. The **pseudoElt** parameter is of type **DOMString**.

Object **DOMImplementationCSS**
DOMImplementationCSS has the all the properties and methods of **DOMImplementation** as well as the properties and methods defined below.
The **DOMImplementationCSS** object has the following methods:
createCSSStyleSheet(title, media)
This method returns a **CSSStyleSheet**. The **title** parameter is of type **DOMString**. The **media** parameter is of type **DOMString**.

Object **ElementCSSInlineStyle**
The **ElementCSSInlineStyle** object has the following properties:
style
This property is of type **CSSStyleDeclaration**.

Object **CSS2Azimuth**
CSS2Azimuth has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.
The **CSS2Azimuth** object has the following properties:
azimuthType
This property is of type **short**.
identifier
This property is of type **String**.
behind
This property is of type **boolean**.
The **CSS2Azimuth** object has the following methods:
setAngleValue(uType, fValue)
This method returns a **void**. The **uType** parameter is of type **unsigned short**. The **fValue** parameter is of type **float**.
getAngleValue(uType)
This method returns a **float**. The **uType** parameter is of type **unsigned short**.
setIdentifier(ident, b)
This method returns a **void**. The **ident** parameter is of type **DOMString**. The **b** parameter is of type **boolean**.

Object **CSS2BackgroundPosition**
CSS2BackgroundPosition has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.
The **CSS2BackgroundPosition** object has the following properties:
horizontalType
This property is of type **short**.

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

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

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

The **CSS2BackgroundPosition** object has the following methods:

getHorizontalPosition(hType)
This method returns a **float**. The **hType** parameter is of type **float**.

getVerticalPosition(vType)
This method returns a **float**. The **vType** parameter is of type **float**.

setHorizontalPosition(hType, value)
This method returns a **void**. The **hType** parameter is of type **unsigned short**. The **value** parameter is of type **float**.

setVerticalPosition(vType, value)
This method returns a **void**. The **vType** parameter is of type **unsigned short**. The **value** parameter is of type **float**.

setPositionIdentifier(hIdentifier, vIdentifier)
This method returns a **void**. The **hIdentifier** parameter is of type **DOMString**. The **vIdentifier** parameter is of type **DOMString**.

Object **CSS2BorderSpacing**

CSS2BorderSpacing has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2BorderSpacing** object has the following properties:

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

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

The **CSS2BorderSpacing** object has the following methods:

getHorizontalSpacing(hType)
This method returns a **float**. The **hType** parameter is of type **float**.

getVerticalSpacing(vType)
This method returns a **float**. The **vType** parameter is of type **float**.

setHorizontalSpacing(hType, value)
This method returns a **void**. The **hType** parameter is of type **unsigned short**. The **value** parameter is of type **float**.

setVerticalSpacing(vType, value)
This method returns a **void**. The **vType** parameter is of type **unsigned short**. The **value** parameter is of type **float**.

Object **CSS2CounterReset**

CSS2CounterReset has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2CounterReset** object has the following properties:

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

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

Object **CSS2CounterIncrement**

CSS2CounterIncrement has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2CounterIncrement** object has the following properties:

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

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

Object **CSS2Cursor**

CSS2Cursor has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2Cursor** object has the following properties:

uris
This property is of type **CSSValueList**.

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

Object **CSS2PlayDuring**

CSS2PlayDuring has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2PlayDuring** object has the following properties:

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

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

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

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

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

Object **CSS2TextShadow**

CSS2TextShadow has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2TextShadow** object has the following properties:

color
This property is of type **CSSValue**.

horizontal
This property is of type **CSSValue**.

vertical
This property is of type **CSSValue**.

blur
This property is of type **CSSValue**.

Object **CSS2FontFaceSrc**

CSS2FontFaceSrc has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2FontFaceSrc** object has the following properties:

uri

This property is of type **String**.

format

This property is of type **CSSValueList**.

fontFaceName

This property is of type **String**.

Object **CSS2FontFaceWidths**

CSS2FontFaceWidths has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2FontFaceWidths** object has the following properties:

urange

This property is of type **String**.

numbers

This property is of type **CSSValueList**.

Object **CSS2PageSize**

CSS2PageSize has the all the properties and methods of **CSSValue** as well as the properties and methods defined below.

The **CSS2PageSize** object has the following properties:

widthType

This property is of type **short**.

heightType

This property is of type **short**.

identifier

This property is of type **String**.

The **CSS2PageSize** object has the following methods:

getWidth(wType)

This method returns a **float**. The **wType** parameter is of type **float**.

getHeightSize(hType)

This method returns a **float**. The **hType** parameter is of type **float**.

setWidthSize(wType, value)

This method returns a **void**. The **wType** parameter is of type **unsigned short**. The **value** parameter is of type **float**.

setHeightSize(hType, value)

This method returns a **void**. The **hType** parameter is of type **unsigned short**. The **value** parameter is of type **float**.

setIdentifier(ident)

This method returns a **void**. The **ident** parameter is of type **DOMString**.

Object **CSS2Properties**

The **CSS2Properties** object has the following properties:

azimuth

This property is of type **String**.

background

This property is of type **String**.

backgroundAttachment
  This property is of type **String**.
backgroundColor
  This property is of type **String**.
backgroundImage
  This property is of type **String**.
backgroundPosition
  This property is of type **String**.
backgroundRepeat
  This property is of type **String**.
border
  This property is of type **String**.
borderCollapse
  This property is of type **String**.
borderColor
  This property is of type **String**.
borderSpacing
  This property is of type **String**.
borderStyle
  This property is of type **String**.
borderTop
  This property is of type **String**.
borderRight
  This property is of type **String**.
borderBottom
  This property is of type **String**.
borderLeft
  This property is of type **String**.
borderTopColor
  This property is of type **String**.
borderRightColor
  This property is of type **String**.
borderBottomColor
  This property is of type **String**.
borderLeftColor
  This property is of type **String**.
borderTopStyle
  This property is of type **String**.
borderRightStyle
  This property is of type **String**.
borderBottomStyle
  This property is of type **String**.
borderLeftStyle
  This property is of type **String**.
borderTopWidth
  This property is of type **String**.

borderRightWidth
This property is of type **String**.
borderBottomWidth
This property is of type **String**.
borderLeftWidth
This property is of type **String**.
borderWidth
This property is of type **String**.
bottom
This property is of type **String**.
captionSide
This property is of type **String**.
clear
This property is of type **String**.
clip
This property is of type **String**.
color
This property is of type **String**.
content
This property is of type **String**.
counterIncrement
This property is of type **String**.
counterReset
This property is of type **String**.
cue
This property is of type **String**.
cueAfter
This property is of type **String**.
cueBefore
This property is of type **String**.
cursor
This property is of type **String**.
direction
This property is of type **String**.
display
This property is of type **String**.
elevation
This property is of type **String**.
emptyCells
This property is of type **String**.
cssFloat
This property is of type **String**.
font
This property is of type **String**.
fontFamily
This property is of type **String**.

fontSize
This property is of type **String**.
fontSizeAdjust
This property is of type **String**.
fontStretch
This property is of type **String**.
fontStyle
This property is of type **String**.
fontVariant
This property is of type **String**.
fontWeight
This property is of type **String**.
height
This property is of type **String**.
left
This property is of type **String**.
letterSpacing
This property is of type **String**.
lineHeight
This property is of type **String**.
listStyle
This property is of type **String**.
listStyleImage
This property is of type **String**.
listStylePosition
This property is of type **String**.
listStyleType
This property is of type **String**.
margin
This property is of type **String**.
marginTop
This property is of type **String**.
marginRight
This property is of type **String**.
marginBottom
This property is of type **String**.
marginLeft
This property is of type **String**.
markerOffset
This property is of type **String**.
marks
This property is of type **String**.
maxHeight
This property is of type **String**.
maxWidth
This property is of type **String**.

minHeight
This property is of type **String**.
minWidth
This property is of type **String**.
orphans
This property is of type **String**.
outline
This property is of type **String**.
outlineColor
This property is of type **String**.
outlineStyle
This property is of type **String**.
outlineWidth
This property is of type **String**.
overflow
This property is of type **String**.
padding
This property is of type **String**.
paddingTop
This property is of type **String**.
paddingRight
This property is of type **String**.
paddingBottom
This property is of type **String**.
paddingLeft
This property is of type **String**.
page
This property is of type **String**.
pageBreakAfter
This property is of type **String**.
pageBreakBefore
This property is of type **String**.
pageBreakInside
This property is of type **String**.
pause
This property is of type **String**.
pauseAfter
This property is of type **String**.
pauseBefore
This property is of type **String**.
pitch
This property is of type **String**.
pitchRange
This property is of type **String**.
playDuring
This property is of type **String**.

position
This property is of type **String**.
quotes
This property is of type **String**.
richness
This property is of type **String**.
right
This property is of type **String**.
size
This property is of type **String**.
speak
This property is of type **String**.
speakHeader
This property is of type **String**.
speakNumeral
This property is of type **String**.
speakPunctuation
This property is of type **String**.
speechRate
This property is of type **String**.
stress
This property is of type **String**.
tableLayout
This property is of type **String**.
textAlign
This property is of type **String**.
textDecoration
This property is of type **String**.
textIndent
This property is of type **String**.
textShadow
This property is of type **String**.
textTransform
This property is of type **String**.
top
This property is of type **String**.
unicodeBidi
This property is of type **String**.
verticalAlign
This property is of type **String**.
visibility
This property is of type **String**.
voiceFamily
This property is of type **String**.
volume
This property is of type **String**.

whiteSpace
This property is of type **String**.
widows
This property is of type **String**.
width
This property is of type **String**.
wordSpacing
This property is of type **String**.
zIndex
This property is of type **String**.

E.6: Document Object Model Events

Object **EventTarget**
The **EventTarget** object has the following methods:
addEventListener(type, listener, useCapture)
This method returns a **void**. The **type** parameter is of type **DOMString**. The **listener** parameter is of type **EventListener**. The **useCapture** parameter is of type **boolean**.
removeEventListener(type, listener, useCapture)
This method returns a **void**. The **type** parameter is of type **DOMString**. The **listener** parameter is of type **EventListener**. The **useCapture** parameter is of type **boolean**.
dispatchEvent(evt)
This method returns a **boolean**. The **evt** parameter is of type **Event**.
Object **EventListener**
This is an ECMAScript function reference. This method returns a **void**. The parameter is of type **Event**.
Class **Event**
The **Event** class has the following constants:
Event.CAPTURING_PHASE
This constant is of type **short** and its value is **1**.
Event.AT_TARGET
This constant is of type **short** and its value is **2**.
Event.BUBBLING_PHASE
This constant is of type **short** and its value is **3**.
Object **Event**
The **Event** object has the following properties:
type
This property is of type **String**.
target
This property is of type **EventTarget**.
currentTarget
This property is of type **EventTarget**.
eventPhase
This property is of type **short**.

bubbles
This property is of type **boolean**.
cancelable
This property is of type **boolean**.
timeStamp
This property is of type **Date**.

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

stopPropagation()
This method returns a **void**.
preventDefault()
This method returns a **void**.
initEvent(eventTypeArg, canBubbleArg, cancelableArg)
This method returns a **void**. The **eventTypeArg** parameter is of type **DOMString**. The **canBubbleArg** parameter is of type **boolean**. The **cancelableArg** parameter is of type **boolean**.

Object **DocumentEvent**

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

createEvent(eventType)
This method returns a **Event**. The **eventType** parameter is of type **DOMString**.

Object **UIEvent**

UIEvent has the all the properties and methods of **Event** as well as the properties and methods defined below.

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

view
This property is of type **AbstractView**.
detail
This property is of type **long**.

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

initUIEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg)
This method returns a **void**. The **typeArg** parameter is of type **DOMString**. The **canBubbleArg** parameter is of type **boolean**. The **cancelableArg** parameter is of type **boolean**. The **viewArg** parameter is of type **views::AbstractView**. The **detailArg** parameter is of type **long**.

Object **MouseEvent**

MouseEvent has the all the properties and methods of **UIEvent** as well as the properties and methods defined below.

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

screenX
This property is of type **long**.
screenY
This property is of type **long**.
clientX
This property is of type **long**.
clientY
This property is of type **long**.

ctrlKey
This property is of type **boolean**.
shiftKey
This property is of type **boolean**.
altKey
This property is of type **boolean**.
metaKey
This property is of type **boolean**.
button
This property is of type **short**.
relatedTarget
This property is of type **EventTarget**.

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

initMouseEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg, screenXArg, screenYArg, clientXArg, clientYArg, ctrlKeyArg, altKeyArg, shiftKeyArg, metaKeyArg, buttonArg, relatedTargetArg)
This method returns a **void**. The **typeArg** parameter is of type **DOMString**. The **canBubbleArg** parameter is of type **boolean**. The **cancelableArg** parameter is of type **boolean**. The **viewArg** parameter is of type **views::AbstractView**. The **detailArg** parameter is of type **long**. The **screenXArg** parameter is of type **long**. The **screenYArg** parameter is of type **long**. The **clientXArg** parameter is of type **long**. The **clientYArg** parameter is of type **long**. The **ctrlKeyArg** parameter is of type **boolean**. The **altKeyArg** parameter is of type **boolean**. The **shiftKeyArg** parameter is of type **boolean**. The **metaKeyArg** parameter is of type **boolean**. The **buttonArg** parameter is of type **unsigned short**. The **relatedTargetArg** parameter is of type **EventTarget**.

Object **MutationEvent**

MutationEvent has the all the properties and methods of **Event** as well as the properties and methods defined below.

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

relatedNode
This property is of type **Node**.
prevValue
This property is of type **String**.
newValue
This property is of type **String**.
attrName
This property is of type **String**.

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

initMutationEvent(typeArg, canBubbleArg, cancelableArg, relatedNodeArg, prevValueArg, newValueArg, attrNameArg)
This method returns a **void**. The **typeArg** parameter is of type **DOMString**. The **canBubbleArg** parameter is of type **boolean**. The **cancelableArg** parameter is of type **boolean**. The **relatedNodeArg** parameter is of type **Node**. The **prevValueArg** parameter is of type **DOMString**. The **newValueArg** parameter is of type **DOMString**. The **attrNameArg** parameter is of type **DOMString**.

The following example will add an ECMA Script based EventListener to the Node ’exampleNode’:

```
// Given the Node ’exampleNode’

// Define the EventListener function
function clickHandler(evt)
{
  // Function contents
}

// The following line will add a non-capturing ’click’ listener
// to ’exampleNode’.
exampleNode.addEventListener("click", clickHandler, false);
```

E.7: Document Object Model Traversal

Object **NodeIterator**

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

root

This property is of type **Node**.

whatToShow

This property is of type **int**.

filter

This property is of type **NodeFilter**.

expandEntityReferences

This property is of type **boolean**.

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

nextNode()

This method returns a **Node**.

previousNode()

This method returns a **Node**.

detach()

This method returns a **void**.

Object **NodeFilter**

This is an ECMAScript function reference. This method returns a **short**. The parameter is of type **Node**.

Object **TreeWalker**

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

root

This property is of type **Node**.

whatToShow

This property is of type **int**.

filter

This property is of type **NodeFilter**.

expandEntityReferences

This property is of type **boolean**.

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

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

parentNode()
This method returns a **Node**.

firstChild()
This method returns a **Node**.

lastChild()
This method returns a **Node**.

previousSibling()
This method returns a **Node**.

nextSibling()
This method returns a **Node**.

previousNode()
This method returns a **Node**.

nextNode()
This method returns a **Node**.

Object **DocumentTraversal**

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

createNodeIterator(root, whatToShow, filter, entityReferenceExpansion)
This method returns a **NodeIterator**. The **root** parameter is of type **Node**. The **whatToShow** parameter is of type **unsigned long**. The **filter** parameter is of type **NodeFilter**. The **entityReferenceExpansion** parameter is of type **boolean**.

createTreeWalker(root, whatToShow, filter, entityReferenceExpansion)
This method returns a **TreeWalker**. The **root** parameter is of type **Node**. The **whatToShow** parameter is of type **unsigned long**. The **filter** parameter is of type **NodeFilter**. The **entityReferenceExpansion** parameter is of type **boolean**.

E.8: Document Object Model Range

Class **Range**

The **Range** class has the following constants:

Range.START_TO_START
This constant is of type **short** and its value is **0**.

Range.START_TO_END
This constant is of type **short** and its value is **1**.

Range.END_TO_END
This constant is of type **short** and its value is **2**.

Range.END_TO_START
This constant is of type **short** and its value is **3**.

Object **Range**

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

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

startOffset
This property is of type **long**.
endContainer
This property is of type **Node**.
endOffset
This property is of type **long**.
collapsed
This property is of type **boolean**.
commonAncestorContainer
This property is of type **Node**.

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

setStart(refNode, offset)
This method returns a **void**. The **refNode** parameter is of type **Node**. The **offset** parameter is of type **long**.
setEnd(refNode, offset)
This method returns a **void**. The **refNode** parameter is of type **Node**. The **offset** parameter is of type **long**.
setStartBefore(refNode)
This method returns a **void**. The **refNode** parameter is of type **Node**.
setStartAfter(refNode)
This method returns a **void**. The **refNode** parameter is of type **Node**.
setEndBefore(refNode)
This method returns a **void**. The **refNode** parameter is of type **Node**.
setEndAfter(refNode)
This method returns a **void**. The **refNode** parameter is of type **Node**.
collapse(toStart)
This method returns a **void**. The **toStart** parameter is of type **boolean**.
selectNode(refNode)
This method returns a **void**. The **refNode** parameter is of type **Node**.
selectNodeContents(refNode)
This method returns a **void**. The **refNode** parameter is of type **Node**.
compareBoundaryPoints(how, sourceRange)
This method returns a **short**. The **how** parameter is of type **unsigned short**. The **sourceRange** parameter is of type **Range**.
deleteContents()
This method returns a **void**.
extractContents()
This method returns a **DocumentFragment**.
cloneContents()
This method returns a **DocumentFragment**.
insertNode(newNode)
This method returns a **void**. The **newNode** parameter is of type **Node**.
surroundContents(newParent)
This method returns a **void**. The **newParent** parameter is of type **Node**.
cloneRange()
This method returns a **Range**.

toString()

This method returns a **DOMString**.

detach()

This method returns a **void**.

Object **DocumentRange**

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

createRange()

This method returns a **Range**.

Acknowledgments

Many people contributed to this specification, including members of the DOM Working Group and the DOM Interest Group. We especially thank the following:

Lauren Wood (SoftQuad Software Inc., *chair*), Arnaud Le Hors (W3C, *W3C staff contact*), Andrew Watson (Object Management Group), Andy Heninger (IBM), Ben Chang (Oracle), Bill Smith (Sun), Bill Shea (Merrill Lynch), Bob Sutor (IBM), 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), James Davidson (Sun), Jared Sorensen (Novell), Joe Kesselman (IBM), Joe Lapp (webMethods), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mike Champion (ArborText and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégaret (W3C), Ramesh Lekshmynarayanan (Merrill Lynch), Ray Whitmer (iMall and Excite@Home), Rich Rollman (Microsoft), Rick Gessner (Netscape), Scott Isaacs (Microsoft), Sharon Adler (INSO), Steve Byrne (JavaSoft), Tim Bray (invited), Tom Pixley (Netscape), Vidur Apparao (Netscape), Vinod Anupam (Lucent).

Thanks to all those who have helped to improve this specification by sending suggestions and corrections.

Thanks to Joe English, author of cost for helping so much in allowing us to develop a framework that made the production of this specification possible.

Thanks to Jan Kärrman, author of html2ps for helping so much in creating the PostScript version of the specification.

Glossary

Editors

Arnaud Le Hors, W3C

Robert S. Sutor, IBM Research (for DOM Level 1)

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.21] . 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.

deepest

The *deepest* element is that element which is furthest from the root or document element in a tree model of the document.

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 `NodeList` [p.47] objects, and their attributes are deeply equivalent.
Two `NodeList` [p.47] objects are *equivalent* if they have the same length, and the nodes corresponding by index are deeply equivalent.
Two `NamedNodeMap` [p.48] 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.68] nodes are *equivalent* if they are equivalent as nodes, have the same names, and have equivalent entities and attributes `NamedNodeMap` [p.48] 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. [HTML4.0]

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

local name

A *local name* is the local part of a *qualified name*. This is called the *local part* in Namespaces in XML [Namespaces].

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.

namespace prefix

A *namespace prefix* is a string that associates an element or attribute name with a *namespace URI* in XML. See *namespace prefix* in Namespaces in XML [Namespaces].

namespace URI

A *namespace URI* is a URI that identifies an XML namespace. This is called the *namespace name* in Namespaces in XML [Namespaces].

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.

qualified name

A *qualified name* is the name of an element or attribute defined as the concatenation of a *local name* (as defined in this specification), optionally preceded by a *namespace prefix* and colon character. See *Qualified Names* in Namespaces in XML [Namespaces].

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. See *Well-Formed XML Documents* in XML [XML].

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.

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].

XML namespace
An *XML namespace* is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. [Namespaces]

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at http://www.w3.org/TR.

H.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

CSS2
: W3C (World Wide Web Consortium) Cascading Style Sheets, level 2 Specification, May 1998. Available at http://www.w3.org/TR/1998/REC-CSS2-19980512

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 8212; Universal Multiple-Octet Coded212; Universal Multiple-Octet Coded Character Set (UCS) 8212; Part 1: Architecture and Basic Multilingual212; 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

Namespaces
: W3C (World Wide Web Consortium) Namespaces in XML, January 1999. Available at http://www.w3.org/TR/1999/REC-xml-names-19990114

OMGIDL
: OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available at http://www.omg.org/library/c2indx.html

RFC2396
: IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt

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, January 2000. Available at http://www.w3.org/TR/1998/REC-xml-19980210

XML-StyleSheet
W3C (World Wide Web Consortium) Associating Style Sheets with XML documents Version 1.0, June 1999. Available at http://www.w3.org/1999/06/REC-xml-stylesheet-19990629.

H.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/library/c2indx.html

DOM-Level-1
W3C (World Wide Web Consortium) DOM Level 1 Specification, October 1998. Available at http://www.w3.org/TR/REC-DOM-Level-1

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/tech/javascript/resources.html

JScript
Microsoft JScript Resources. Available at http://msdn.microsoft.com/scripting/default.htm

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

Objects Index

Document Object Model Core

Document Object Model HTML

Document Object Model Views

Document Object Model StyleSheets

Index