WD-DOM-Level-2-19990304

Document Object Model (DOM) Level 2 Specification

Version 1.0

W3C Working Draft *04 March, 1999*

This version
http://www.w3.org/TR/1999/WD-DOM-Level-2-19990304
http://www.w3.org/TR/1999/WD-DOM-Level-2-19990304/DOM2.ps
http://www.w3.org/TR/1999/WD-DOM-Level-2-19990304/DOM2.pdf
http://www.w3.org/TR/1999/WD-DOM-Level-2-19990304/DOM2.txt
http://www.w3.org/TR/1999/WD-DOM-Level-2-19990304/DOM2.zip
Latest version
http://www.w3.org/TR/WD-DOM-Level-2
Previous versions
http://www.w3.org/TR/1998/WD-DOM-Level-2-19981228

WG Chair
Lauren Wood, *SoftQuad Software Inc.*
Editors
Vidur Apparao, *Netscape Communications Corporation*
Mike Champion, *Arbortext and Aliaron*
Arnaud Le Hors, *W3C*
Tom Pixley, *Netscape Communications Corporation*
Jonathan Robie, *Texcel Research*
Peter Sharpe, *SoftQuad Software Inc.*
Chris Wilson, *Microsoft*
Lauren Wood, *SoftQuad Software Inc.*

Status of this document

This document is an early release of the Document Object Model Level 2. It is guaranteed to change; anyone implementing it should realize that we will not allow ourselves to be restricted by experimental implementations of Level 2 when deciding whether to change the specifications.

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

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

This document is for public review. . Comments on this document should be sent to the public mailing list www-dom@w3.org.

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. Level 2 is expected to add interfaces for a Cascading Style Sheets object model, an event model, and a query interface, amongst others.

This release of the Document Object Model Level 2 does not have all of the interfaces that the final version will have. It contains interfaces for associating stylesheets with a document, the Cascading Style Sheets object model, the Range object model, filters and iterators, and the Events object model. The DOM WG wants to get feedback on the interfaces that are in this version of the DOM Level 2 specification. The other interfaces will be added in future versions of this specification.

Table of contents

Expanded Table of Contents

Copyright Notice

Copyright Notice

1. Document Object Model (Core) Level 2

Editors
Arnaud Le Hors, W3C

1.1. Overview of the DOM Level 2 Core Interfaces

This section will define an additional set of interfaces that augment the interfaces defined in the Core section of the DOM Level 1 Recommendation to provide functionalities which are found to be essential but were not addressed in the Level 1.

These functionalitites could be:

- Creating a `Document` [p.57] object
- Moving a node from one document to another
- Equality and hashcodes
- A way to determine the document order of nodes
- Creating EntityReference nodes
- Conversion of a CDATASection node to a TEXT node
- A way to decorate a node with client data
- A way to get the element an attribute is attached to
- A way to join two adjacent Text nodes

2. Document Object Model Namespaces

Editors
Arnaud Le Hors, W3C

2.1. Introduction

This section will define a new set of interfaces that augment the interfaces defined in the Core section to deal with XML namespaces .

3. Document Object Model StyleSheets

Editors

Vidur Apparao, Netscape Communications Corp.
Chris Wilson, Microsoft

3.1. 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.

3.2. Style Sheet Interfaces

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

Interface *StyleSheet*

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

```
interface StyleSheet {
  readonly attribute  DOMString            type;
           attribute  boolean              disabled;
  readonly attribute  Node                 owningNode;
  readonly attribute  StyleSheet           parentStyleSheet;
  readonly attribute  DOMString            href;
  readonly attribute  DOMString            title;
  readonly attribute  DOMString            media;
};
```

Attributes

`type`
: 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 `ownindNode`. A list of registered content types can be found at ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/ . 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 .

`disabled`
: `false` if the style sheet is applied to the document. `true` if it is not.

`owningNode`
: 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 included style sheets, this attribute has a value of null.

`parentStyleSheet`
: 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 the attribute is null.

`href`
: 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 .

`title`
: The advisory title. The title is often specified in the `owningNode`. 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 .

`media`
: The intended destination medium for style information. It may be a single media descriptor or a comma-separated list. The media is often specified in the `owningNode`. 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 .

Interface *StyleSheetCollection*

The `StyleSheetCollection` interface provides the abstraction of an ordered collection of style sheets.

IDL Definition

```
interface StyleSheetCollection {
  readonly attribute  unsigned long        length;
  StyleSheet                item(in unsigned long index);
};
```

Attributes

`length`
: The length or the size of the list.

Methods

`item`
: Used to retrieve a style sheet by ordinal index.

Parameters

`index`	Index into the collection

Return Value

The style sheet at the `index` position in the `StyleSheetCollection`, or `null` if that is not a valid index.

This method raises no exceptions.

4. Document Object Model CSS

Editors
Vidur Apparao, Netscape Communications Corp.
Chris Wilson, Microsoft

4.1. 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 two). 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 Style Sheets chapter of the DOM Level 2. 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 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 interface.

The most common type of rule is a style declaration. The CSSStyleRule 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 interface.

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

4.2. CSS Fundamental Interfaces

The interfaces within this section are considered fundamental, and must be implemented by all conforming applications of this specifcation. These interfaces represent CSS style sheets specifically.

Interface *CSSStyleSheet*

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

```
interface CSSStyleSheet : StyleSheet {
  readonly attribute  CSSRuleCollection    cssRules;
  unsigned long             insertRule(in DOMString rule,
                                       in unsigned long index)
                                       raises(DOMException);
  void                      deleteRule(in unsigned long index)
                                       raises(DOMException);
};
```

Attributes

`cssRules`

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

Methods

`insertRule`

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

Parameters

`rule`	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`	The index within the style sheet’s rule collection 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

The index within the style sheet’s rule collection of the newly inserted rule.

Exceptions

`DOMException`

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.

SYNTAX_ERR: Raised if the specified rule has a syntax error and is unparsable.

`deleteRule`

Used to delete a rule from the style sheet.

Parameters

`index`	The index within the style sheet’s rule collection of the rule to remove.

Exceptions

`DOMException`

INDEX_SIZE_ERR: Raised if the specified index does not correspond to a rule in the style sheet’s rule collection.

This method returns nothing.

Interface *CSSRuleCollection*

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

IDL Definition

```
interface CSSRuleCollection {
  readonly attribute  unsigned long        length;
  CSSRule                   item(in unsigned long index);
};
```

Attributes

`length`

The length or the size of the list.

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`	Index into the collection

Return Value

The style rule at the `index` position in the `CSSRuleCollection`, or `null` if that is not a valid index.

This method raises no exceptions.

Interface *CSSRule*

The `CSSRule` interface is the abstract base interface for any type of CSS statement . This includes both rule sets and at-rules .

IDL Definition

```
interface CSSRule {
  // RuleType
  const unsigned short      UNKNOWN_RULE       = 0;
  const unsigned short      STYLE_RULE         = 1;
  const unsigned short      IMPORT_RULE        = 2;
  const unsigned short      MEDIA_RULE         = 3;
  const unsigned short      FONT_FACE_RULE     = 4;
  const unsigned short      PAGE_RULE          = 5;

  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

UNKNOWN_RULE	The rule is a `CSSUnknownRule` [p.24] .
STYLE_RULE	The rule is a `CSSStyleRule` [p.21] .
IMPORT_RULE	The rule is a `CSSImportRule` [p.23] .
MEDIA_RULE	The rule is a `CSSMediaRule` [p.22] .
FONT_FACE_RULE	The rule is a `CSSFontFaceRule` [p.23] .
PAGE_RULE	The rule is a `CSSPageRule` [p.23] .

Attributes

`type`
A code defining the type of the rule, as defined above.

`cssText`
The parsable textual representation of the rule.
Exceptions on setting
`DOMException`

SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable.

`parentStyleSheet`
The style sheet that contains this rule.

`parentRule`
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`.

Interface *CSSStyleRule*

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

```
interface CSSStyleRule : CSSRule {
           attribute  DOMString            selectorText;
  readonly attribute  CSSStyleDeclaration  style;
};
```

Attributes

`selectorText`
The textual representation of the selector for the rule set. The implementation may have stripped out insignificant whitespace while parsing the selector.

`style`
The declaration-block of this rule set.

Interface *CSSMediaRule*

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

```
interface CSSMediaRule : CSSRule {
           attribute  DOMString            mediaTypes;
  readonly attribute  CSSRuleCollection    cssRules;
  unsigned long             insertRule(in DOMString rule,
                                       in unsigned long index)
                                       raises(DOMException);
  void                      deleteRule(in unsigned long index);
};
```

Attributes

`mediaTypes`

A comma-separate list of media types for this rule. This attribute does not include the `"@media"` specifier.

`cssRules`

A collection of all CSS rules contained within the media block.

Methods

`insertRule`

Used to insert a new rule into the media block.

Parameters

`rule`	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`	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

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

Exceptions

`DOMException`

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.

SYNTAX_ERR: Raised if the specified rule has a syntax error and is unparsable.

`deleteRule`

Used to delete a rule from the media block.

Parameters

`index`	The index within the media block's rule collection of the rule to remove.

This method returns nothing.
This method raises no exceptions.

Interface *CSSFontFaceRule*

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

```
interface CSSFontFaceRule : CSSRule {
  readonly attribute  CSSStyleDeclaration  style;
};
```

Attributes

`style`

The declaration-block of this rule.

Interface *CSSPageRule*

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

```
interface CSSPageRule : CSSRule {
           attribute  DOMString            selectorText;
  readonly attribute  CSSStyleDeclaration  style;
};
```

Attributes

`selectorText`

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

`style`

The declaration-block of this rule.

Interface *CSSImportRule*

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

```
interface CSSImportRule : CSSRule {
           attribute  DOMString            href;
           attribute  DOMString            media;
  readonly attribute  CSSStyleSheet        styleSheet;
};
```

Attributes

`href`

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

`media`

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

`styleSheet`

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 *CSSUnknownRule*

The `CSSUnkownRule` interface represents an at-rule not supported by this user agent.

IDL Definition

```
interface CSSUnknownRule : CSSRule {
};
```

Interface *CSSStyleDeclaration*

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.

IDL Definition

```
interface CSSStyleDeclaration {
           attribute  DOMString            cssText;
                                       // raises(DOMException) on setting
  DOMString                getPropertyValue(in DOMString propertyName);
  DOMString                removeProperty(in DOMString propertyName);
  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`

The parsable textual representation of the declaration block (including 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.

Exceptions on setting

`DOMException`

SYNTAX_ERR: Raised if the specified CSS string value has a syntax error and is unparsable.

Methods

`getPropertyValue`

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

Parameters

`propertyName`	The name of the CSS property. See the CSS property index .

Return Value

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.

This method raises no exceptions.

`removeProperty`

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

Parameters

`propertyName`	The name of the CSS property. See the CSS property index .

Return Value

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 valid CSS2 property.

This method raises 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`	The name of the CSS property. See the CSS property index .

Return Value

A string representing the priority (e.g. `"important"`) if one exists. The empty string if none exists.

This method raises no exceptions.

`setProperty`

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

Parameters

`propertyName`	The name of the CSS property. See the CSS property index .
`value`	The new value of the property.
`priority`	The new priority of the property (e.g. `"important"`).

Exceptions

`DOMException`

SYNTAX_ERR: Raised if the specified value has a syntax error and is unparsable.

This method returns nothing.

Attributes

`length`

The number of properties that have been explicitly set in this declaration block.

Methods

`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`	Index of the property name to retrieve.

Return Value

The name of the property at this ordinal position. The empty string if no property exists at this position.

This method raises no exceptions.

Attributes

`parentRule`

The CSS rule that contains this declaration block.

4.3. CSS Extended Interfaces

The interfaces found within this section are not mandatory. They may be implemented by a DOM implementation as a convenience to the DOM script user.

Interface *CSS2Properties*

The `CSS2Properties` interface represents a convenience mechanism for retrieving and setting properties within a `CSSStyleDeclaration` [p.24] . 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` [p.24] interface. Setting an attribute of this interface is equivalent to calling the `setProperty` method of the `CSSStyleDeclaration` [p.24] interface.

A compliant implementation is not required to implement 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".

IDL Definition

```
interface CSS2Properties {
           attribute  DOMString           azimuth;
           attribute  DOMString           background;
           attribute  DOMString           backgroundAttachment;
           attribute  DOMString           backgroundColor;
           attribute  DOMString           backgroundImage;
           attribute  DOMString           backgroundPosition;
           attribute  DOMString           backgroundRepeat;
           attribute  DOMString           border;
           attribute  DOMString           borderCollapse;
           attribute  DOMString           borderColor;
           attribute  DOMString           borderSpacing;
           attribute  DOMString           borderStyle;
           attribute  DOMString           borderTop;
           attribute  DOMString           borderRight;
           attribute  DOMString           borderBottom;
           attribute  DOMString           borderLeft;
           attribute  DOMString           borderTopColor;
           attribute  DOMString           borderRightColor;
           attribute  DOMString           borderBottomColor;
           attribute  DOMString           borderLeftColor;
           attribute  DOMString            borderTopStyle;
           attribute  DOMString            borderRightStyle;
           attribute  DOMString            borderBottomStyle;
           attribute  DOMString            borderLeftStyle;
           attribute  DOMString            borderTopWidth;
           attribute  DOMString            borderRightWidth;
           attribute  DOMString            borderBottomWidth;
           attribute  DOMString            borderLeftWidth;
           attribute  DOMString            borderWidth;
           attribute  DOMString            bottom;
           attribute  DOMString            captionSide;
           attribute  DOMString            clear;
           attribute  DOMString            clip;
           attribute  DOMString            color;
           attribute  DOMString            content;
           attribute  DOMString            counterIncrement;
           attribute  DOMString            counterReset;
           attribute  DOMString            cue;
           attribute  DOMString            cueAfter;
           attribute  DOMString            cueBefore;
           attribute  DOMString            cursor;
           attribute  DOMString            direction;
           attribute  DOMString            display;
           attribute  DOMString            elevation;
           attribute  DOMString            emptyCells;
           attribute  DOMString            cssFloat;
           attribute  DOMString            font;
           attribute  DOMString            fontFamily;
           attribute  DOMString            fontSize;
           attribute  DOMString            fontSizeAdjust;
           attribute  DOMString            fontStretch;
           attribute  DOMString            fontStyle;
           attribute  DOMString            fontVariant;
           attribute  DOMString            fontWeight;
           attribute  DOMString            height;
           attribute  DOMString            left;
           attribute  DOMString            letterSpacing;
           attribute  DOMString            lineHeight;
           attribute  DOMString            listStyle;
           attribute  DOMString            listStyleImage;
           attribute  DOMString            listStylePosition;
           attribute  DOMString            listStyleType;
           attribute  DOMString            margin;
           attribute  DOMString            marginTop;
           attribute  DOMString            marginRight;
           attribute  DOMString            marginBottom;
           attribute  DOMString            marginLeft;
           attribute  DOMString            markerOffset;
           attribute  DOMString            marks;
           attribute  DOMString            maxHeight;
           attribute  DOMString            maxWidth;
           attribute  DOMString            minHeight;
           attribute  DOMString            minWidth;
           attribute  DOMString            orphans;
           attribute  DOMString            outline;
           attribute  DOMString            outlineColor;
           attribute  DOMString            outlineStyle;
```

```
          attribute  DOMString            outlineWidth;
          attribute  DOMString            overflow;
          attribute  DOMString            padding;
          attribute  DOMString            paddingTop;
          attribute  DOMString            paddingRight;
          attribute  DOMString            paddingBottom;
          attribute  DOMString            paddingLeft;
          attribute  DOMString            page;
          attribute  DOMString            pageBreakAfter;
          attribute  DOMString            pageBreakBefore;
          attribute  DOMString            pageBreakInside;
          attribute  DOMString            pause;
          attribute  DOMString            pauseAfter;
          attribute  DOMString            pauseBefore;
          attribute  DOMString            pitch;
          attribute  DOMString            pitchRange;
          attribute  DOMString            playDuring;
          attribute  DOMString            position;
          attribute  DOMString            quotes;
          attribute  DOMString            richness;
          attribute  DOMString            right;
          attribute  DOMString            size;
          attribute  DOMString            speak;
          attribute  DOMString            speakHeader;
          attribute  DOMString            speakNumeral;
          attribute  DOMString            speakPunctuation;
          attribute  DOMString            speechRate;
          attribute  DOMString            stress;
          attribute  DOMString            tableLayout;
          attribute  DOMString            textAlign;
          attribute  DOMString            textDecoration;
          attribute  DOMString            textIndent;
          attribute  DOMString            textShadow;
          attribute  DOMString            textTransform;
          attribute  DOMString            top;
          attribute  DOMString            unicodeBidi;
          attribute  DOMString            verticalAlign;
          attribute  DOMString            visibility;
          attribute  DOMString            voiceFamily;
          attribute  DOMString            volume;
          attribute  DOMString            whiteSpace;
          attribute  DOMString            widows;
          attribute  DOMString            width;
          attribute  DOMString            wordSpacing;
          attribute  DOMString            zIndex;
};
```

Attributes

`azimuth`

See the azimuth property definition in CSS2.

`background`

See the background property definition in CSS2.

`backgroundAttachment`

See the background-attachment property definition in CSS2.

`backgroundColor`
: See the background-color property definition in CSS2.

`backgroundImage`
: See the background-image property definition in CSS2.

`backgroundPosition`
: See the background-position property definition in CSS2.

`backgroundRepeat`
: See the background-repeat property definition in CSS2.

`border`
: See the border property definition in CSS2.

`borderCollapse`
: See the border-collapse property definition in CSS2.

`borderColor`
: See the border-color property definition in CSS2.

`borderSpacing`
: See the border-spacing property definition in CSS2.

`borderStyle`
: See the border-style property definition in CSS2.

`borderTop`
: See the border-top property definition in CSS2.

`borderRight`
: See the border-right property definition in CSS2.

`borderBottom`
: See the border-bottom property definition in CSS2.

`borderLeft`
: See the border-left property definition in CSS2.

`borderTopColor`
: See the border-top-color property definition in CSS2.

`borderRightColor`
: See the border-right-color property definition in CSS2.

`borderBottomColor`
: See the border-bottom-color property definition in CSS2.

`borderLeftColor`
: See the border-left-color property definition in CSS2.

`borderTopStyle`
: See the border-top-style property definition in CSS2.

`borderRightStyle`
: See the border-right-style property definition in CSS2.

`borderBottomStyle`
: See the border-bottom-style property definition in CSS2.

`borderLeftStyle`
: See the border-left-style property definition in CSS2.

`borderTopWidth`
: See the border-top-width property definition in CSS2.

`borderRightWidth`
: See the border-right-width property definition in CSS2.

`borderBottomWidth`
: See the border-bottom-width property definition in CSS2.

`borderLeftWidth`
: See the border-left-width property definition in CSS2.

`borderWidth`
: See the border-width property definition in CSS2.

`bottom`
: See the bottom property definition in CSS2.

`captionSide`
: See the caption-side property definition in CSS2.

`clear`
: See the clear property definition in CSS2.

`clip`
: See the clip property definition in CSS2.

`color`
: See the color property definition in CSS2.

`content`
: See the content property definition in CSS2.

`counterIncrement`
: See the counter-increment property definition in CSS2.

`counterReset`
: See the counter-reset property definition in CSS2.

`cue`
: See the cue property definition in CSS2.

`cueAfter`
: See the cue-after property definition in CSS2.

`cueBefore`
: See the cue-before property definition in CSS2.

`cursor`
: See the cursor property definition in CSS2.

`direction`
: See the direction property definition in CSS2.

`display`
: See the display property definition in CSS2.

`elevation`
: See the elevation property definition in CSS2.

`emptyCells`
: See the empty-cells property definition in CSS2.

`cssFloat`
: See the float property definition in CSS2.

`font`
: See the font property definition in CSS2.

`fontFamily`
: See the font-family property definition in CSS2.

`fontSize`
: See the font-size property definition in CSS2.

`fontSizeAdjust`
: See the font-size-adjust property definition in CSS2.

`fontStretch`
: See the font-stretch property definition in CSS2.

`fontStyle`
: See the font-style property definition in CSS2.

`fontVariant`
: See the font-variant property definition in CSS2.

`fontWeight`
: See the font-weight property definition in CSS2.

`height`
: See the height property definition in CSS2.

`left`
: See the left property definition in CSS2.

`letterSpacing`
: See the letter-spacing property definition in CSS2.

`lineHeight`
: See the line-height property definition in CSS2.

`listStyle`
: See the list-style property definition in CSS2.

`listStyleImage`
: See the list-style-image property definition in CSS2.

`listStylePosition`
: See the list-style-position property definition in CSS2.

`listStyleType`
: See the list-style-type property definition in CSS2.

`margin`
: See the margin property definition in CSS2.

`marginTop`
: See the margin-top property definition in CSS2.

`marginRight`
: See the margin-right property definition in CSS2.

`marginBottom`
: See the margin-bottom property definition in CSS2.

`marginLeft`
: See the margin-left property definition in CSS2.

`markerOffset`
: See the marker-offset property definition in CSS2.

`marks`
: See the marks property definition in CSS2.

`maxHeight`
: See the max-height property definition in CSS2.

`maxWidth`
: See the max-width property definition in CSS2.

`minHeight`
: See the min-height property definition in CSS2.

`minWidth`
: See the min-width property definition in CSS2.

`orphans`
: See the orphans property definition in CSS2.

`outline`
: See the outline property definition in CSS2.

`outlineColor`
: See the outline-color property definition in CSS2.

`outlineStyle`
: See the outline-style property definition in CSS2.

`outlineWidth`
: See the outline-width property definition in CSS2.

`overflow`
: See the overflow property definition in CSS2.

`padding`
: See the padding property definition in CSS2.

`paddingTop`
: See the padding-top property definition in CSS2.

`paddingRight`
: See the padding-right property definition in CSS2.

`paddingBottom`
: See the padding-bottom property definition in CSS2.

`paddingLeft`
: See the padding-left property definition in CSS2.

`page`
: See the page property definition in CSS2.

`pageBreakAfter`
: See the page-break-after property definition in CSS2.

`pageBreakBefore`
: See the page-break-before property definition in CSS2.

`pageBreakInside`
: See the page-break-inside property definition in CSS2.

`pause`
: See the pause property definition in CSS2.

`pauseAfter`
: See the pause-after property definition in CSS2.

`pauseBefore`
: See the pause-before property definition in CSS2.

`pitch`
: See the pitch property definition in CSS2.

`pitchRange`
: See the pitch-range property definition in CSS2.

`playDuring`
: See the play-during property definition in CSS2.

`position`
: See the position property definition in CSS2.

`quotes`
: See the quotes property definition in CSS2.

`richness`
: See the richness property definition in CSS2.

`right`
: See the right property definition in CSS2.

`size`
: See the size property definition in CSS2.

`speak`
: See the speak property definition in CSS2.

`speakHeader`
: See the speak-header property definition in CSS2.

`speakNumeral`
: See the speak-numeral property definition in CSS2.

`speakPunctuation`
: See the speak-punctuation property definition in CSS2.

`speechRate`
: See the speech-rate property definition in CSS2.

`stress`
: See the stress property definition in CSS2.

`tableLayout`
: See the table-layout property definition in CSS2.

`textAlign`
: See the text-align property definition in CSS2.

`textDecoration`
: See the text-decoration property definition in CSS2.

`textIndent`
: See the text-indent property definition in CSS2.

`textShadow`
: See the text-shadow property definition in CSS2.

`textTransform`
: See the text-transform property definition in CSS2.

`top`
: See the top property definition in CSS2.

`unicodeBidi`
: See the unicode-bidi property definition in CSS2.

`verticalAlign`
: See the vertical-align property definition in CSS2.

`visibility`
: See the visibility property definition in CSS2.

`voiceFamily`
: See the voice-family property definition in CSS2.

`volume`
: See the volume property definition in CSS2.

`whiteSpace`
: See the white-space property definition in CSS2.

`widows`
: See the widows property definition in CSS2.

`width`
: See the width property definition in CSS2.

`wordSpacing`
: See the word-spacing property definition in CSS2.

`zIndex`
: See the z-index property definition in CSS2.

4.4. Extensions to Level 1 Interfaces

(**ED:** This section will dissipate into other sections of the Level 2 DOM as they develop. These extensions are placed here until those other sections are prepared.)

4.4.1. Document style sheets

A collection of all style sheets linked into or embedded in the document is exposed through the `styleSheets` attribute. In HTML, this collection contains both external style sheets, included via the LINK element, and inline style sheets, included via STYLE elements. In XML, this collection contains all external style sheets included via a style sheet processing instruction .

```
interface  Document2 : Document {
  readonly attribute StyleSheetCollection styleSheets;
};
```

4.4.2. HTMLElement inline style

Inline style information attached to HTML elements is exposed through the `style` attribute. This represents the contents of the STYLE attribute for HTML elements.

```
interface HTMLElement2 : HTMLElement {
  readonly attribute CSSStyleDeclaration style;
};
```

4.4.3. HTMLStyleElement style sheet

The style sheet associated with an HTML STYLE element is accessible via the styleSheet attribute.

```
interface HTMLStyleElement2 : HTMLStyleElement {
  readonly attribute StyleSheet styleSheet;
};
```

4.4.4. HTMLLinkElement style sheet

The styleSheet associated with an HTML LINK element with a REL of "stylesheet" or "alternate stylesheet" is not accessible directly. This is because LINK elements are not used purely as a stylesheet linking mechanism. The `styleSheet` property on LINK elements with other relationships would be incongruous.

4.5. Unresolved Issues

1. The CSS Editorial team is considering a way to represent comments that exist within a CSS style sheet. Our expectation is that absolute position of comments may not be maintained, but relative position (with respect to CSS rules and CSS properties) and the actual contents of the comment will be.
2. The CSS Editorial team is considering a mechanism to allow users to retrieve the cascaded and computed styles for a specific element.We do not intend to provide access to the actual style of specific elements in this level of the CSS DOM. Implementation of the CSS DOM does not require an actual rendering engine for any other reason, and we see that requirement as a limitation on the potential implementations of the CSS DOM.
3. The CSS Editorial team is considering a mechanism to allow users to change the cascaded style for a specific element, or to create rules in an "override" style sheet.
4. The Working Group is still considering whether it should be possible to create style sheets outside the context of a document, abstract from any XML- or HTML-specific embedding or linking of a style sheet.
5. The DOM Working Group is considering whether more structure is necessary in the representation of CSS style rules; specifically, providing access to the selector other than as a string, and providing more structured representation of the varied CSS2 properties.
6. The group is undecided whether to put a cssText attribute on the CSSStyleSheet, which would provide a textual representation of the entire style sheet. Setting this attribute would result in the resetting of all the rules in the style sheet.
7. We still need to create a CSSException inherited from DOMException, to allow easier catching of CSS-specific exceptions.

5. Document Object Model Events

Editors
Tom Pixley, Netscape Communications Corporation
Chris Wilson, Microsoft Corporation

5.1. 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 attempt to 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 within Microsoft Internet Explorer 4.0 and Netscape Navigator 4.0. 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.

5.1.1. 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.

Cancellable
: 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.

5.1.2. Requirements

The following constitutes the list of requirements for the DOM Level 2 Event Model.
(**ED:** Not all of the requirements below are addressed in the current version of the specification. However, all of the requirements which derive from existing event systems should currently be met.)

Requirements of event flow:

- The model must support multiple event listeners on a single `Node`.
- The model must support the ability to receive events both before and after the DOM implementation has processed the event allowing the action which triggered the event to take place.

Requirements of event listener registration:

- The model must define a programmatic mechanism of specifying event listeners. This mechanism must be rich enough to support custom events, chaining of multiple event listeners, and general event listener registration
- If additional methods of registering event listeners are defined they must be consistent with the programmatic model for event listener registration. Consistent means it is possible to define a sequence of DOM API calls which would have the same result.
- The model must define the interaction between the programmatic event registration mechanism and event listener registration within HTML tags defined in the HTML 4.0 Specification
- The programmatic method of event listener registration should allow the client to specify whether to receive the event before or after it has been processed by the DOM implementation.
- Tag based registration, style based registration, and programmatic registration must all be able to coexist together. The event model must define rules for interaction between them.

Requirements of contextual event information:

- The model must specify a mechanism for providing basic contextual information for any event.
- The model must specify a mechanism to provide UI events with additional UI specific information.

Requirements of event types:

- The model must allow the creation of additional event sets beyond those specified within the DOM Level 2 Event Model specification.
- The model must support UI events.
- The model must define a set of UI logical events to allow reaction to UI input in a device independent way. One use of this is for accessibility.
- The model must define a set of document mutation events which allow notification of any change to the document’s structure.
- The model should define a set of events to allow notification of changes to a document’s style.

5.2. 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 target `Node` level or centrally from a `Node` higher in the document tree.

5.2.1. Basic event flow

Each event has a `Node` toward which the event is directed by the DOM implementation. This `Node` is the event target. When the event reaches the target, any event listeners registered on the `Node` are triggered. Although all `EventListener` [p.43] s on the `Node` are guaranteed to receive the event, no specification is made as to the order in which they will receive the event with regards to the other `EventListener` [p.43] s on the `Node`. 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.

5.2.2. Event Capture

Event capture is the process by which an ancestor of the event's target can register to intercept events of a given type before they are received by the event's target. Capture operates from the top of the tree downward, making it the symmetrical opposite of bubbling which is described below.

An `EventListener` [p.43] being registered on an `EventTarget` [p.41] may choose to have that `EventListener` [p.43] 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 either no additional capturing `EventListener` [p.43] s are found or the event's target is reached.

If the capturing `EventListener` [p.43] wishes to prevent further processing of the event it may set the `cancelCapture` property of the `Event` [p.44] to true. This will prevent further dispatch of the event to additional `EventTarget`s lower in the tree structure, although additional `EventListener` [p.43] s registered at the same hierarchy level will still receive the event. However, if after dispatching the event to the final `EventListener` [p.43] at a given level, the value of `cancelCapture` is false, the implementation then propagates the event down to the next capturing `EventListener` [p.43] existing between itself and the event's target. If no additional capturers exist, the event triggers the appropriate `EventListener` [p.43] s on the target itself.

Although event capture is similar to the delegation based event model, it is different in two important respects. First, event capture only allows interception of events which are targeted at descendants of the capturing `Node`. 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 `Node`, 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.

5.2.3. 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 their target `Node` and any event listeners found there are triggered. Bubbling events then perform a check of the event's `cancelBubble` attribute. If the attribute is false, the event will then look for additional event listeners by following the `Node`'s parent chain upward, checking for any event listeners registered on each successive `Node`. This upward propagation will continue all the way up to the `Document` [p.57] unless either the bubbling process is prevented through use of the `cancelBubble` attribute.

An event handler may choose to prevent continuation of the bubbling process at any time through use of the `cancelBubble` attribute on the event object. After dispatching the event to all `EventListener` [p.43] s on a given `EventTarget` [p.41] the value of the `cancelBubble` property is checked. If the value is true, bubbling will cease at that level. If the value is false, bubbling will continue upward to the parent of the current `EventTarget` [p.41] .

5.2.4. Event cancellation

Some events are specified as cancellable. For these events, the DOM implementation generally has a default action associated with the event. 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 cancelling the implementation's default action or allowing the default action to proceed. Cancellation is accomplished by setting the event's `returnValue` attribute to false.

5.3. Event listener registration

5.3.1. Event registration interfaces

Interface *EventTarget*

The `EventSource` interface is implemented by `Node`s which can be targetted by events. The interface allows event listeners to be registered on the node.

IDL Definition

```
interface EventTarget {
  void                      addEventListener(in DOMString type,
                                             in boolean postProcess,
                                             in boolean useCapture,
                                             in EventListener listener);
  void                      removeEventListener(in DOMString type,
                                                in boolean postProcess,
                                                in boolean useCapture,
                                                in EventListener listener);
};
```

Methods

`addEventListener`

This method allows the registration of event listeners on the event target.

Parameters

`type`	The event type for which the user is registering
`postProcess`	If true, `postProcess` indicates that the user wishes to register to receive events after any action associated with the event has occurred. If the value is false, it indicates the user wishes to receive the event before any action has occurred. Some events can only be handled using one or the other of these techniques. (**ED:** Should an invalid postProcess value raise an exception?)
`useCapture`	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` [p.43] before being dispatched to any `EventTarget`s beneath them in the tree. Events which are bubbling upward through the tree will not trigger an `EventListener` [p.43] designated to use capture.
`listener`	The `listener` parameter takes an interface implemented by the user which contains the methods to be called when the event occurs.

This method returns nothing.
This method raises no exceptions.

`removeEventListener`

This method allows the removal of event listeners from the event target. If an `EventListener` [p.43] is removed from an `EventTarget` while it is processing an event, it will complete its current actions but will not be triggered again during any later stages of event flow.

Parameters

`type`	Specifies the event type of the `EventListener` [p.43] being removed.
`postProcess`	Specifies whether the `EventListener` [p.43] being removed is a preProcess or postProcess listener.
`useCapture`	Specifies whether the `EventListener` [p.43] being removed is a capturing listener or not.
`listener`	The `EventListener` [p.43] parameter indicates the `EventListener` to be removed.

This method returns nothing.
This method raises no exceptions.

Interface *EventListener*

The `EventListener` interface is the primary method for handling events. Users implement the `EventListener` interface and register their listener on a `EventTarget` using the `AddEventListener` method.

IDL Definition

```
interface EventListener {
  void                      handleEvent(in Event event);
};
```

Methods

`handleEvent`

This method is called whenever an event occurs of the type for which the `EventListener` interface was registered.

Parameters

`event`	The `Event` [p.44] contains contextual information about the event. It also contains the `returnValue` and `cancelBubble` properties which are used in determining proper event flow.

This method returns nothing.
This method raises no exceptions.

5.3.2. Interaction with HTML 4.0 event listeners

In HTML 4.0, event listeners where specified as properties of an element. As such, registration of a second event listeners of the same type would override the value of the first listener. The DOM Event Model allows registration of multiple event listeners on a single `Node`. To achieve this, event listeners are no longer stored as property values.

In order to achieve compatibility with HTML 4.0, implementors may view the setting of properties which represent event handlers as the creation and registration of an `EventListener` on the `Node`. The value for `postProcess` should be given a default value appropriate for the event. This `EventListener` [p.43] behaves in the same manner as any other `EventListeners`s which may be registered on the `Node`. If the property representing the event listener is changed, this may be viewed as the removal of the previously registered `EventListener` [p.43] and the registration of a new one.

5.3.3. Event listener registration issues

The first issue is a question of whether listeners should exist as typed interfaces containing groups of similar events or instead as a single generic listener. An example of the first case would be:

```
interface MouseListener : EventListener{
        MouseDown();
        MouseUp();
        Click();
}
```

whereas the second is:

```
interface EventListener {
        HandleEvent();
}
```

The specification currently defines listeners via the second solution. This solution avails itself more readily to extending or creating new events. The first solution would require defintion of new event interfaces in order to add events. However, remaining problems with the first solution include the fact that registering the same object for multiple events requires the user to differentiate between the events inside the event listener. The current string based event typing system could make this very inefficient. The DOM Working Group is exploring alternatives to the string based event typing to resolve this issue.

The second issue concerns event ordering. If multiple event handlers are registered on the same node ordering may need to be imposed on the event delivery. One solution to this includes adding an ordering scheme into the listener registration mechanism. This would also necessitate adding a method for introspection of registered listeners to `EventTarget` . A second solution imposes ordering through registration order. However, this breaks down quickly if multithreading is allowed. A third solution is to specify that event ordering is left to the application.

Lastly, a full solution has not yet been added to meet the suggestion that all listeners be notified of the final resolution of an event. It is possible that use of both pre- and post-processing of events will achieve this goal but it is not yet clear if this solution will be sufficient.

5.4. Event interfaces

Interface *Event*

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

```
interface Event {
          attribute  DOMString            type;
          attribute  Node                 target;
          attribute  Node                 currentNode;
          attribute  boolean              cancelBubble;
          attribute  boolean              cancelCapture;
          attribute  boolean              returnValue;
};
```

Attributes

`type`
: The `type` property represents the event name as a string property.

`target`
: The `target` property indicates the `Node` to which the event was originally dispatched.

`currentNode`
: The `currentNode` property indicates to which `Node` the event is currently being dispatched. This is particularly useful during capturing and bubbling.

`cancelBubble`
: The `cancelBubble` property is used to control the bubbling phase of event flow. If the property is set to true, the event will cease bubbling at the current level. If the property is set to false, the event will bubble up to its parent. The default value of this property is determined by the event type.

`cancelCapture`
: The `cancelCapture` property is used to control propagation during the capturing phase of event flow. If the property is set to true, the event will not propagate down any further in the tree. If the property is set to false, the event will continue down to the next capturing node, or if none exists, to the event target. The default value of this property is false.

`returnValue`
: If an event is cancellable, the `returnValue` property is checked by the DOM implementation after the event has been processed by its event handlers. If the `returnValue` is false, the DOM implementation does not execute any default actions associated with the event.

Interface *UIEvent*

The `UIEvent` interface provides specific contextual information associated with User Interface and Logical events.

IDL Definition

```
interface UIEvent : Event {
          attribute  long                 screenX;
          attribute  long                 screenY;
          attribute  long                 clientX;
          attribute  long                 clientY;
          attribute  boolean              altKey;
          attribute  boolean              ctrlKey;
          attribute  boolean              shiftKey;
          attribute  unsigned long        keyCode;
          attribute  unsigned long        charCode;
          attribute  unsigned short       button;
};
```

Attributes

`screenX`
: `screenX` indicates the horizontal coordinate at which the event occurred in relative to the origin of the screen coordinate system.

`screenY`
: `screenY` indicates the vertical coordinate at which the event occurred relative to the origin of the screen coordinate system.

`clientX`
: `clientX` indicates the horizontal coordinate at which the event occurred relative to the DOM implementation’s client area.

`clientY`
: `clientY` indicates the vertical coordinate at which the event occurred relative to the DOM implementation’s client area.

`altKey`
: `altKey` indicates whether the ’alt’ key was depressed during the firing of the event.

`ctrlKey`
: `ctrlKey` indicates whether the ’ctrl’ key was depressed during the firing of the event.

`shiftKey`
: `shiftKey` indicates whether the ’shift’ key was depressed during the firing of the event.

`keyCode`
: The value of `keyCode` holds the virtual key code value of the key which was depressed if the event is a key event. Otherwise, the value is zero.

`charCode`
: `charCode` holds the value of the Unicode character associated with the depressed key if the event is a key event. Otherwise, the value is zero.

`button`
: During mouse events caused by the depression or release of a mouse button, `button` is used to indicate which mouse button changed state.

Interface *MutationEvent*

The `MutationEvent` interface provides specific contextual information associated with Mutation events.

IDL Definition

```
interface MutationEvent : Event {
           attribute  Node                 relatedNode;
           attribute  DOMString            prevValue;
           attribute  DOMString            newValue;
           attribute  DOMString            attrName;
};
```

Attributes

`relatedNode`
: `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.

`prevValue`
: `prevValue` indicates the previous value of text nodes and attributes in attrModified and charDataModified events.

`newValue`
: `newValue` indicates the new value of text nodes and attributes in attrModified and charDataModified events.

`attrName`
: `attrName` indicates the changed attr in the attrModified event.

5.4.1. Event object issues

The main issue with respect to the `Event` [p.44] object regards how this object will be made accessible to the `EventListener` . The specification current passes the `Event` [p.44] as the first parameter of the `handleEvent` method. However, some compatibility concerns have been raised with this approach. Alternatives to this method are being explored.

A secondary issue exists regarding the possible addition of a new property to the base `Event` [p.44] interface to indicate to which `Node` the event is currently being dispatched. This would alleviate possible confusion during the bubbling and capturing phases when the same `EventListener` [p.43] is registered upon multiple nodes. The property has been added while its necessity is under discussion.

5.5. 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, a set of UI logical events, and a set of document mutation events.

5.5.1. 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 both Netscape Navigator 4.0 and Microsoft Internet Explorer 4.0.

User Inteface event issues: Different implementations receive user interface events in different orders or don’t receive all events specified. For example, in some implemenations a dblclick event might occur as the user presses the mouse button down, in others it may occur as the user releases the mouse button. There are two possible solutions to this. The first is that the DOM Level 2 Events specification my define the user interface events that will be delivered and the order in which they will be delivered. Implementations would then deliver the events specified, making translations as necessary from the events being delivered to the implementation. The other solution is to define User Interface events as varying from implementation to implemenation, making no guarantee on the ordering of event delivery.

click
: The click event occurs when the pointing device button is clicked over an element. This attribute may be used with most elements.
  - Bubbles: Yes

- Cancellable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, button

dblclick

The dblclick event occurs when the pointing device button is double clicked over an element. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, button

mousedown

The mousedown event occurs when the pointing device button is pressed over an element. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, button

mouseup

The mouseup event occurs when the pointing device button is released over an element. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, button

mouseover

The mouseover event occurs when the pointing device is moved onto an element. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey

mousemove

The mousemove event occurs when the pointing device is moved while it is over an element. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: No
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey

mouseout

The mouseout event occurs when the pointing device is moved away from an element. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey

keypress

The keypress event occurs when a key is pressed and released. This attribute may be used with most elements.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: keyCode, charCode

keydown
: The keydown event occurs when a key is pressed down. This attribute may be used with most elements.
  - Bubbles: Yes
  - Cancellable: Yes
  - Context Info: keyCode, charCode

keyup
: The keyup event occurs when a key is released. This attribute may be used with most elements.
  - Bubbles: Yes
  - Cancellable: Yes
  - Context Info: keyCode, charCode

resize
: The resize event occurs when a document is resized.
  - Bubbles: Yes
  - Cancellable: No
  - Context Info: None

scroll
: The scroll event occurs when a document is scrolled.
  - Bubbles: Yes
  - Cancellable: No
  - Context Info: None

5.5.2. 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 cancellable. The reasoning for this stems from the fact that it would be 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 cancellation 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.

It should also be noted that many of the mutation events have been designed in pairs, one which bubbles and one which does not. An example of this is the pair of events childInsertedOntoParent and nodeInsertedOntoParent. The first event, childInsertedOntoParent, is dispatched to the prospective parent node and bubbled up through the document. The second event is dispatched to the child node and does not bubble. The intention is that both the child and parent will be able to receive the desired notifications whether registered as pre-processing or post-processing `EventListener` [p.43] s. For example, when an `EventListener` [p.43] is registered for pre-processing of this event, the child `Node` is not yet attached to its new parent and bubbling is insufficient to allow notification of the imminent structural change to both the child and parent. Thus, pairs of events are necessary to describe all possible document changes. One of each pair of these events is designated as non-bubbling to prevent overlapping notifications when handling the post-processing listener case.

subtreeModified

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. Also, the requirement for some sort of batching of mutation events may be accomplished through this event. The target of this event is the lowest common parent of the changes which have taken place.

- Bubbles: Yes
- Cancellable: No
- Context Info: None

nodeInsertedOntoParent

Fired when a node is added as a child of another node. The target of this event is the node being inserted.

- Bubbles: No
- Cancellable: No
- Context Info: relatedNode holds the parent node

nodeRemovedFromParent

Fired when a node is removed from another node. The target of this event is the node being removed.

- Bubbles: No
- Cancellable: No
- Context Info: relatedNode holds the parent node

childInsertedOntoParent

Fired when a node is added as a child of another node. The target of this event is the parent onto which the node was inserted.

- Bubbles: Yes
- Cancellable: No
- Context Info: relatedNode holds the child node

childRemovedFromParent

Fired when a node is removed from another node. The target of this event is the parent from which the child was removed.

- Bubbles: Yes
- Cancellable: No
- Context Info: relatedNode holds the child node

nodeRemovedFromDocument

Fired when a node is removed from a document. The target of this event is the node being removed.

- Bubbles: No
- Cancellable: No
- Context Info: None

nodeInsertedIntoDocument

Fired when a node is inserted into a document. The target of this event is the node being inserted.

- Bubbles: No
- Cancellable: No
- Context Info: None

nodeRemovedFromSubtree

Fired when a node is removed from a subtree. The target of this event is the top of the subtree.

- Bubbles: Yes
- Cancellable: No

- Context Info: relatedNode holds the removed node

nodeInsertedIntoSubtree

Fired when a node is inserted into a subtree. The target of this event is the top of the subtree.

- Bubbles: Yes
- Cancellable: No
- Context Info: relatedNode holds the inserted node

attrModified

Fired when an `attr` is modified on a node. The target of this event is the node whose `attr` changed.

- Bubbles: Yes
- Cancellable: No
- Context Info: attrName, prevValue, newValue

characterDataModified

Fired when `CharacterData` within a node is modified but the node itself has not been inserted or deleted. The target of this event is the `CharacterData` node.

- Bubbles: Yes
- Cancellable: No
- Context Info: prevValue, newValue

5.5.3. HTML event types

The HTML event set is composed of events listed in HTML 4.0 and additional events which are supported in both Netscape Navigator 4.0 and Microsoft Internet Explorer 4.0.

load

The load event occurs when the DOM implementation finishes loading all content within a document, all frames within a FRAMESET, or an image.

- Bubbles: No
- Cancellable: No
- Context Info: None

unload

The unload event occurs when the DOM implementation removes a document from a window or frame. This attribute may be used with BODY and FRAMESET elements.

- Bubbles: No
- Cancellable: No
- Context Info: None

abort

The abort event occurs when page loading is stopped before an image has been allowed to completely load. This attribute applies to the IMG element.

- Bubbles: Yes
- Cancellable: 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 attribute applies to the IMG element and to the BODY and FRAMESET element.

- Bubbles: Yes
- Cancellable: No
- Context Info: None

select

The select event occurs when a user selects some text in a text field. This attribute may be used with the INPUT and TEXTAREA elements.

- Bubbles: Yes
- Cancellable: 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 attribute applies to the following elements: INPUT, SELECT, and TEXTAREA.

- Bubbles: Yes
- Cancellable: No
- Context Info: None

submit

The submit event occurs when a form is submitted. It only applies to the FORM element.

- Bubbles: Yes
- Cancellable: Yes
- Context Info: None

reset

The reset event occurs when a form is reset. It only applies to the FORM element.

- Bubbles: Yes
- Cancellable: No
- Context Info: None

focus

The focus event occurs when an element receives focus either via a pointing device or by tabbing navigation. This attribute may be used with the following elements: LABEL, INPUT, SELECT, TEXTAREA, and BUTTON.

- Bubbles: No
- Cancellable: No
- Context Info: None

blur

The blur event occurs when an element loses focus either by the pointing device or by tabbing navigation. It may be used with the same elements as onfocus

- Bubbles: No
- Cancellable: No
- Context Info: None

6. Document Object Model Filters and Iterators

Editors

Mike Champion, Aliaron

Jonathan Robie, Texcel

6.1. Overview of the DOM Level 2 Query, Iterator, and Filter Interfaces

The DOM Level 2 Query, Iterator, and Filter interfaces extend the functionality of the DOM to allow simple and efficient traversal of document subtrees, node lists, or the results of queries.

This proposal contains Iterator and Filter interfaces, but no query interfaces. A separate specification will be prepared for query interfaces, which will be query-language independent.

6.1.1. Iterators

In several popular approaches to software design, iterators are considered a basic building block for building reusable software and software libraries. For instance, they are fundamental to the Design Patterns approach, STL, and the Java libraries. The main advantages of node iterators in the DOM are:

1. Abstracting out the way that specific data structures are navigated. Functions that use iterators can operate on any data structure without knowing the details of how that data structure is navigated; e.g., the same function could process the nodes in a document, a document subtree, or a nodelist. The function can keep asking for the next node without worrying about how that node is found.
2. Allowing more efficient navigation. Because an iterator hides the manner in which a data structure is navigated, it can use indexes or other supplementary data structures to allow more efficient navigation than might be possible by naively navigating from one node to the next.
3. Providing views for the most common ways applications want to navigate document structures. Some applications traverse only the element tree, others process additional nodes such as processing instructions or comments, others prefer yet another view. There is no one right way to navigate a document tree, but iterators provide a simple, efficient way to choose the most appropriate view of the document tree for a given application.

An iterator allows the nodes of a data structure to be returned sequentially. When an iterator is first created, calling nextNode() returns the first node. When no more nodes are present, nextNode() returns a null. It is important to remember that DOM structures may change as a document is loaded - when nextNode() finds no more nodes, it is still quite possible that further nodes may be added in the next instant. Since iterators do not know how to predict the future, there is no way to check whether further nodes may be added at any given time.

Since the DOM permits liveness and editing, and an iterator may be active while the data structure it navigates is being edited, an iterator must behave gracefully in the face of change. Additions and deletions in the underlying data structure do not invalidate an iterator.

Using ordered set semantics, the position of the iterator is determined by the relative position in the ordered set. There is no current node. When an iterator is created for a list, the position is set before the first element:

```
 A B C D E F G H I
^
```

Each call to next() returns a node and advances the position. For instance, if we start with the above position, the first call to next() returns "A" and advances the iterator:

```
 A B C D E F G H I
  ^
```

The relative position of the iterator remains valid when nodes are deleted. Suppose the nodes in our list do not come from a tree, but are merely a set of nodes in which none of the nodes are children of other nodes. If you delete "A", the position of the iterator is unchanged with respect to the remaining nodes:

```
 B C D E F G H I
^
```

Similarly, if "B" and "C" are deleted, the position remains unchanged with respect to the remaining nodes:

```
 D E F G H I
^
```

Moving the "D" node to the end of the set does not change the current position:

```
 E F G H I D
^
```

Note that the relative position of the iterator is not the same as the absolute position within the set. The position of the iterator is relative to the node before it and the node after it, which is why the position floats gracefully when nodes are deleted or inserted before or after the position of the iterator. If an iterator were based on absolute position, then an iterator at position 5 would suddenly point to a different item if node 3 were deleted. In many implementations, iterators may need to be adjusted when nodes are inserted or deleted.
(**ED:** The fix-ups required by this model complicate implementation somewhat, but make life simpler for the user of iterators. Much of the complexity of fix-ups is in notification - the fix-ups themselves are then relatively straightforward. It might seem simpler to invalidate an iterator when changes are made, but invalidation also requires notification. We currently feel that handling change gracefully is worth the added implementation cost, but are interested in feedback on this issue.)

6.1.2. Filters

Filters allow the user to "filter out" nodes. Each filter contains a user-written function that looks at a node and determines whether or not it should be filtered out. To use a filter, you create an iterator that uses the filter. The iterator applies the filter to each node, and if the filter rejects the node, the iterator skips over the node as though it were not present in the document. Filters are easy to write, since they need not know how to navigate the structure on which they operate, and they can be reused for different kinds of iterators that operate on different data structures.

Let’s use a filter to write code to find the named anchors in an HTML document. In HTML, an HREF can refer to any <A> element that has a NAME attribute. The first step is to write a filter that looks at a node and determines whether it is a named anchor:

```
class NamedAnchorFilter implements NodeFilter
{
    boolean acceptNode(Node n) {
        if (n instanceof Element) {
            Element e = n;
            if (n.getAttribute("NAME") != NULL) {
                return true;
            }
        }
        return false;
    }
}
```

To use this filter, create an instance of the filter and create an iterator using it:

These flags can be combined using OR:

```
Node iter=factory.create(root, TW_ELEMENT | TW_PI | TW_COMMENT | TW_EXPANDED);
```

The default view shows elements and text, but no other nodes (attributes are retrieved from the elements). The constant TW_DEFAULT is a mask that defines this default view.

If TW_ENTITYREF is not set, entities are expanded. If TW_ENTITYREF is set, entity references will be encountered by the iterator. There is no setting that shows both the entity reference and its expansion. (**ED:** We need to specify the details of how this will work in ECMAScript, which does not have the concept of abstract interfaces or data types, more formally)

```
NamedAnchorFilter naf;
NodeIterator nit = document.createFilteredTreeIterator(naf);
```

At this point, the iterator will show only the named anchors in the document. Writing equivalent code without filters would be marginally simpler, and no less efficient. The advantage of using filters is that it allows reuse. For instance, if you have another part of your program that needs to find the named anchors in a NodeList, you can use the filter the same way you used it for the document:

```
NamedAnchorFilter naf;
NodeIterator nit = nodelist.createFilteredTreeIterator(naf);
```

6.2. Formal Interface Definition

Interface *NodeIterator*

NodeIterators are used to step through a set of nodes, e.g. the set of nodes in a NodeList, the document subtree governed by a particular node, the results of a query, or any other set of nodes. The set of nodes to be iterated is determined by the factory that creates the iterator.

Any iterator that returns nodes may implement the NodeIterator interface. Users and vendor libraries may also choose to create iterators that implement the NodeIterator interface.

IDL Definition

```
interface NodeIterator {
  Node                      nextNode();
  Node                      prevNode();
};
```

Methods

`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

The next `Node` in the set being iterated over, or NULL if there are no more members in that set.

This method has no parameters.

This method raises no exceptions.

`prevNode`

Returns the previous node in the set and moves the position of the iterator backwards in the set.

Return Value

The previous `Node` in the set being iterated over, or NULL if there are no more members in that set.

This method has no parameters.

This method raises no exceptions.

(**ED:** Some felt that firstNode() and lastNode() would be useful to position to the beginning or end of the iterated set. Others felt this requires the implementation to maintain too much state. For now, we have chosen not to specify these methods, but we are open to feedback on this issue. One implementor suggested that prevNode() was too complex when nodes are kept in a singly linked list. We suspect that the ability to traverse in both directions is extremely useful, and a quick, informal poll suggested that most DOM implementations probably need to do this already.)

Interface *Document*

Document contains methods that creates iterators 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).

IDL Definition

```
interface Document {
  boolean                   createTreeIterator(in Node root,
                                               in short whatToShow);
};
```

(**ED:** What about createListIterator?)

(**ED:** In a later version of Level 2, when queries are supported, we will also want factory methods that can issue a query and provide an iterator for the result set. These methods may look something like this:

```
    NodeIterator createTreeQueryIterator(DOMString query);
    NodeIterator createListQueryIterator(DOMString query);
```

)

Methods

`createTreeIterator`

Parameters

`root`	The node which will be iterated together with its children.
`whatToShow`	This flag determines whether entities are expanded, and whether comments, processing instructions, or text are presented via the iterator.

```
public static final int  TW_DEFAULT       = 0x0022;
public static final int  TW_ALL           = 0xFFFF;
public static final int  TW_ELEMENT       = 0x0002;
public static final int  TW_PI            = 0x0008;
public static final int  TW_COMMENT       = 0x0010;
public static final int  TW_TEXT          = 0x0020;
public static final int  TW_ENTITYREF     = 0x0040;
```

These flags can be combined using OR:

```
    Node iter=factory.create(root, TW_ELEMENT | TW_PI | TW_COMMENT | TW_EXPANDED);
```

The default view shows elements and text, but no other nodes (attributes are retrieved from the elements). The constant TW_DEFAULT is a mask that defines this default view.

If TW_ENTITYREF is not set, entities are expanded. If TW_ENTITYREF is set, entity references will be encountered by the iterator. There is no setting that shows both the entity reference and its expansion.
(**ED:** Several people have suggested that the functionality of whatToShow be implemented using filters. We feel that it is better to implement them using iterators, since it makes it possible to provide a more efficient implementation. A filter must examine each node individually; an iterator can make use of internal data structures to examine only those nodes that are desired.)

Return Value

TRUE if a this node is to be passed through the filter and returned by the `NodeIterator::nextNode()` method, FALSE if this node is to be ignored.

This method raises no exceptions.

Interface *NodeFilter*

Filters are simply objects that know how to "filter out" nodes. If an iterator is given a filter, before it returns the next node, it applies the filter. If the filter says to accept the node, the iterator returns it; otherwise, the iterator looks for the next node and pretends that the node that was rejected was not there.

The DOM does not provide any filters. Filter is just an interface that users can implement to provide their own filters. The introduction to this chapter gives an example of how a user can implement a filter to perform a specific function.

Filters do not need to know how to iterate, nor do they need to know anything about the data structure that is being iterated. 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 iterators, encouraging code reuse.

IDL Definition

```
interface NodeFilter {
  boolean                   acceptNode(in Node n);
};
```

Methods

`acceptNode`

Parameters

`n`	The node to check to see if it passes the filter or not.

Return Value

TRUE if a this node is to be passed through the filter and returned by the `NodeIterator::nextNode()` method, FALSE if this node is to be ignored.

This method raises no exceptions.

7. Document Object Model Range

Editors
Vidur Apparao, Netscape Communications
Peter Sharpe, SoftQuad Software Inc.

7.1. Introduction

The Range object identifies a single contiguous sequence of content in a document (or document fragment). It can be thought of as a pair of end points which define the boundary of the content ’selected’ by the range. The term ’selected’ does not mean that every range appears to a user as a GUI selection, however such a GUI selection can be returned to a DOM user via a Range.

The Range object provides methods for accessing and manipulating the document tree at a higher level than the related Node object methods. This proposal defines the basic functionality, that is, how to create and move a Range object and how to use Ranges to insert, delete and copy content. It is anticipated that a future version of the Range object will include further convenience functions which would be of use to authors using the DOM.

7.1.1. Motivation

The Range object is useful for several reasons:

First, it will be useful to be able to retrieve the user’s selection -- for example in response to events -- and perform actions on that selection.

Second, the Range object provides editing and querying functionality on a range in the document, rather than on a node basis as is possible with Node objects . For example, the ubiquitous cut, copy and paste editing operations are expected to work on a contiguous group of nodes. It is possible to implement these operations using the primitive Node editing operations, but it requires looping and testing whereas the same functionality can be accomplished by a single Range method call.

And third, it will be extremely common to apply editing operations to a range of the document, and a Range can be useful for locking that range when we come to supporting concurrent update.

In summary, the Range object conveniently packages up editing and querying operations on ranges in a document whereas the Node and NodeList objects are restricted to single nodes.

7.1.2. Basic Assumptions

The Range object approximately corresponds to a range in the raw document with the end-points of the range on token boundaries. This means that an end-point of the Range cannot be in the middle of a start- or end-tag, or within an entity reference (in the raw structure model) or the replacement entity itself in the cooked structure model. The Range object locates a contiguous portion of the content of the structure model.

It must be possible for a Range to select across element boundaries. Results of this must be defined carefully for each operation on the Range.

In terms of the DOM object hierarchy, the Range object has no base object. In particular, it is not derived from Node. Unless otherwise stated, all methods in this section are methods of the Range object.

7.1.3. Notation

Most of the examples in the proposal will be illustrated using the text representation of a document. The portion of the document selected by a range will be shown in bold text as in

<FOO>A**BC<BAR>DE**F</BAR></FOO>

When the selected portion contains no content (both endpoints are at the same position) it will be shown as a bold caret ('^') as in

<FOO>A**^**BC<BAR>DEF</BAR></FOO>

And when referring to a single end-point, it will be show as a bold asterisk ('*****') as in

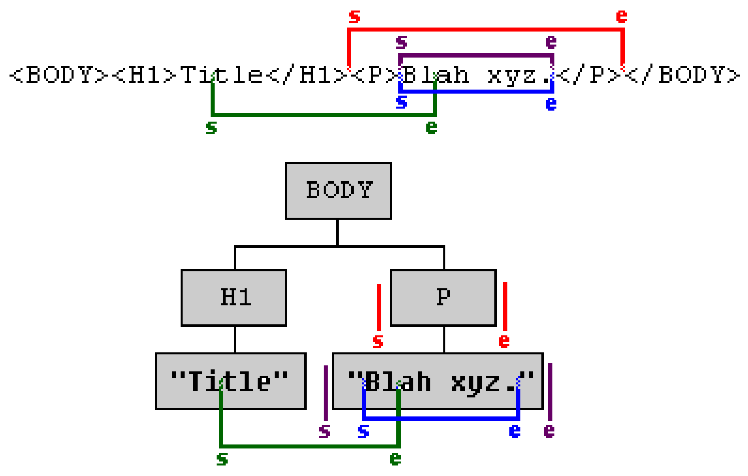
<FOO>A*****BC<BAR>DEF</BAR></FOO>

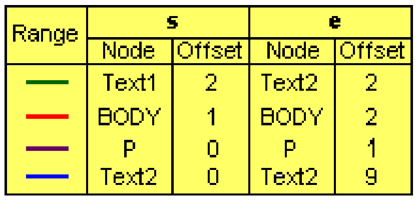
7.2. Finding a Range's Position

A Range has two end-points (the start and the end). Each end-point's position in a document (or document fragment) can be characterized by two quantities: a parent node and an offset relative to that parent node. The Range is considered to select the contiguous content of the document or document fragment contained between the two end-points.

Note that a Range only selects within the document tree. In particular, the parent node of a Range's end-point must be an Element, Comment, ProcessingInstruction, EntityReference, CDATASection, Document, DocumentFragment or Text node and it must have a Document or DocumentFragment node as an ancestor. This requirement specifically excludes Attr, DocumentType, Entity and Notation nodes as ancestors of end-point parents.
(**ED:** The Working Group is considering allowing Attr nodes as ancestors of end-point parents with the restriction that both end-points have the same Attr node as an ancestor. This would allow range operations on an attribute tree in the same manner as on a document tree.)

The relationship between locations in the raw source document and in the Node tree interface of the DOM is illustrated in the following diagram:





Range	s		e	
	Node	Offset	Node	Offset
🟩	Text1	2	Text2	2
🟥	BODY	1	BODY	2
🟪	P	0	P	1
🟦	Text2	0	Text2	9

Range Example

In this diagram, four different Ranges are illustrated. Consider the red Range with end-points labelled s and e. This Range selects the entire P node.

In the raw source, it is possible and convenient to specify the location of the end-points by using absolute offsets from the beginning of the document. In this case, the red Range could be said to select the content of the raw source document from after the 20th character to after the 36th character.

There are several reasons why absolute offsets are not a useful way to specify end-points in the DOM tree. First of all, such absolute offsets are potentially very inefficient to calculate and maintain. Second, two different end-points in the tree can have the same absolute offset in the raw document as will be discussed below. And, finally, since they refer to the persisted state of the document, calculating the offsets would require the DOM to precisely specify how the document is persisted.

For these reasons, the end-points are specified using a node and an offset within the children of that node. In the example above, the position represented by the end-point labelled s is within the BODY element. It is after the H1 element and before the P element so it corresponds to a position between the H1 and P children of BODY. The offset of an end-point within its containing node is 0 if it is before the first child, 1 if between the first and second child, and so on. So, for end-point s, the container node is BODY and the offset is 1. For end-points within text nodes, the offset is specified similarly but using character positions instead. For example, the end-point labelled s has a Text node as its container and an offset of 2 since it is between the second and third characters.

The diagram and table illustrates the container nodes and offsets for the end-points of four Ranges. Notice that the corresponding end-points of purple and blue ranges appear to be identical in the raw document but that each is, in fact, represented distinctly in the DOM. This is an important feature of the Range since it means that an end-point of a Range can unambiguously represent every position within the document tree.

When the parent node of an end-point is not a text node, the offset specifies a position between the child nodes. For example, an offset of 0 means that the end-point is before the first child, an offset of 1 means it is after the first child and before the second child, and so on.

However, it is also often convenient to think of a Range as selecting a portion of the raw source document and many of the examples in this specification will be illustrated that way.

The parents and offsets of the end-points can be accessed using the following read-only Range attributes:

```
startParent;
startOffset;
endParent;
endOffset;
```

If both end-points of a Range have the same parent nodes and offsets then the Range is a degenerate selection, or collapsed Range. (This is often referred to as an insertion point in a user agent.)

7.3. Partial and Complete Containment

A node is said to be partially contained by a Range if it is an ancestor of or equal to the containing node of one or both end-points of the Range. That is, if the node contains at least one end of the Range, then it is partially contained. For example, consider the green Range in Diagram 1, above. H1 is partially contained by that Range since the start end-point is within one of its children. And BODY is partially contained by the same Range since both end-points are contained within children of its children.

A node is said to be completely contained by a Range if it is located between the the two end-points of the Range. In terms of the raw source document, a node would only be completely contained by a Range if its corresponding start-tag was located after the starting end-point of the Range and its end-tag was located before the end of the Range. In the examples in Diagram 1, above, the red Range completely contains the P node and the purple Range completely contains the text node containing the text "Blah xyz."

7.4. Creating a Range

(**ED:** The factory method for creating a Range should be implemented by the document object. Since this involves a new method, it may either be added to the existing Document interface or a secondary interface implemented by the same object. The determination of where this method goes and how to deal with new methods on existing interfaces in backwardly compatible manner needs to be addressed by the Working Group as a whole.)

A range is created by calling a method on the Document object:

```
interface Document {
  interface Document {
  ....
  Range createRange();
}
```

The initial state of the range returned from this method is such that its two end-points are equal and both are positioned at the beginning of the Document before any content. In other words, the parent node of each end-point is the Document node and the offset within that node is 0.

Like some other objects created from the Document (like Nodes and DocumentFragments), Ranges created via a particular document instance are only compatible with content associated with that document, and cannot be used with other document instances.

7.5. Changing a Range's Position

A Range's position can be specified by setting the parent and offset of each end-point with the `setStart` and `setEnd` methods.

If one end-point of a Range is set to be positioned in content associated with a document fragment other than that in which the range is currently positioned, the range will be collapsed to the new location. This enforces the restriction that both end-points of a Range must be in the same document or fragment.

Also, the start position is guaranteed to never be to the right of the end position. As a consequence of this, attempting to set the start to be to the right of the end will cause the end to be moved to the same position, resulting in a collapsed range at that location. The case for the end being before the start is similarly handled.

It is also possible to set a Range's position relative to other nodes in the tree:

```
void setStartBefore( in Node sibling );
void setStartAfter( in Node sibling );
void setEndBefore( in Node sibling );
void setEndAfter( in Node sibling );
```

The parent of the sibling node will become the parent of the end-point and the Range will be subject to the same restrictions as outlined above for setStart() and setEnd().

A Range can be collapsed to either end-point:

```
void collapse ( in boolean toStart );
```

Passing TRUE to the parameter toStart will collapse the range to the range's start position, FALSE to the end.

Testing if a Range is collapsed can be done by examining the isCollapsed attribute:

```
readonly attribute boolean isCollapsed;
```

Quite often one will want to cause a range to select everything under a node, possibly including the node itself:

```
void selectNode ( in Node n );
void selectNodeContents ( in Node n );
```

For example:

```
Before:
  ^<BAR><FOO>A<MOO>B</MOO>C</FOO></BAR>
After range.selectNodeContents( FOO ):
  <BAR><FOO>A<MOO>B</MOO>C</FOO></BAR>
After range.selectNode( FOO ):
  <BAR><FOO>A<MOO>B</MOO>C</FOO></BAR>
```

7.6. Comparing Range End-Points

It is possible to compare two Ranges by comparing their end-points:

```
int compareEndPoints(CompareHow how, Range sourceRange)
```

where CompareHow is one of 4 values: StartToStart, StartToEnd, EndToEnd and EndToStart. The return value is -1, 0 or 1 depending on whether the corresponding end-point of the Range is less than, equal or greater than the corresponding end-point of sourceRange.

Determining if one end-point is less than another requires examing a number of cases but, informally, one end-point is less than another if it corresponds to a location in the source document before the second end-point. This can be stated more precisely in terms of the DOM tree, as follows:

If both end-points have the same parent node, then one end-point is less than the other if its offset is less the offset of the other end-point.

If the end-points have different parent nodes, then there are three cases to consider.

Let A and B be the two end-points. The first case to consider is when a child of the parent of A is the parent or an ancestor of the parent of B. In this case, A is less than B if the offset of A is less than or equal to the index of the child containing B.

The second case is when a child of the parent of B is the parent or an ancestor of the parent of A. In this case, A is less than B if the index of the child containing A is less than the offset of B.

The third case is when neither parent is an ancestor of the other end-point's parent. In this case, let N be the common ancestor of both A and B which has the greatest depth in the DOM tree. Then A is less than B if the index of the child of N which is an ancestor of the parent of A is less than the index of the child of N which is an ancestor of the parent of B.

Comparing two end-points for equality is much more straightforward: Two end-points are equal to one another if and only if they have the same parents and both offsets are equal.

And finally, determining if one end-point is greater than another can be stated in terms of the other two comparisons: A is greater than B if A is not equal to B and A is not less than B.

Note that because the same location in the source document can correspond to two different locations in the DOM tree, it is possible for two end-points to not compare equal even though they would be equal in the source. For this reason, the informal definition above can sometimes be misleading.

7.7. Deleting Content with a Range

One can delete the contents selected by a range with:

```
void deleteContents ( );
```

The deletion of the contents selected by a range is pretty straight forward if the parent nodes for each endpoint is the same. For example:

```
<FOO><MOO>CD</MOO></FOO>  -->  <FOO>^</FOO>
```

Here, the range has endpoints (each endpoint expressed as a pair Node, Offset) of (FOO, 0) and (FOO, 1). Notice in this example that the MOO node was removed in its entirety. This is so because the MOO began and ended within the scope of the range’s selection. Thus, any node which starts and ends within a range’s selection is removed in its entirety. Also notice that the FOO tag was left untouched (other than its immediate content being modified). Thus, any node which starts and ends outside a range’s selection is not affected.

There are two other cases left to completely describe the effect on a document of the deleteContents operation:

```
1) <FOO>A<MOO>BC</MOO>DE</FOO>  -->  <FOO>A<MOO>B</MOO>^E</FOO>

2) <FOO>XY<BAR>ZW</BAR>Q</FOO>  -->  <FOO>X^<BAR>W</BAR>Q</FOO>
```

In case 1, the MOO node begins before the range’s selection, while the MOO’s end is contained within the ranges selection. Here, it is important to know that the deleteContents operation is structural, not textual. Stated differently, the deleteContents operation on a range does not remove the textual representation of its content, as though one were editing the document contents (including tags) in a text editor. While, as in this example, the textual representation of the range selection may include only one of the start- or end-tag representing an element, a deleteContents operation on that range will not result in a non-well formed document.

A node is considered to be "partially" contained within a range if, in the textual representation of the range, only one of either its start- or end-tag is included in the range contents. In this case, a deleteContents operation will not remove the partially contained element. However, after the operation is completed, the (now collapsed) range will move outside the element. Specifically, if the range’s original start point were before the node (in depth-first post-order) the range would collapse to a position before the node. If the range’s original end point were after the node, the range would collapse to a position after the node.

```
<FOO>A<MOO>B^E</FOO>
```

Now, notice that in this, false, example there is a begin tag for the MOO node, but no end tag. This is not representable by the DOM. All nodes in the DOM must have a definite begin and end. Thus, notice how the end tag of the MOO node effectively scooted to the left, outside the influence of the range’s selection. This is so because only a part of the MOO node was deleted. If the begin of the MOO node was inside the selection of the range at the time of the deletion, then the MOO node would have been removed in it entirety. For case 2, instead of the later half of a node falling within the range, the first half is contained within the range. This is very similar to case 1, with the exception that the begin tag for BAR scoots to the right.

To summarize these two cases where only a part of a node is selected, if the node begins in the selection, the begin tag, effectively, scoots to the right, if the node ends in the selection, the end tag, effectively, scoots left.

In cases where the contents of a range should be extracted rather than deleted, the following method may be used:

```
DocumentFragment extractContents ( );
```

The extractContents method does exactly what the deleteContents methods does, but it additionally places the deleted contents in a new DocumentFragment. Using the three examples above, the following illustrate the contents of the returned document fragment:

```
<FOO><MOO>CD</MOO></FOO>     -->  <MOO>CD</MOO>

<FOO>A<MOO>BC</MOO>DE</FOO>  -->  <MOO’>C</MOO’>D   (MOO’ is a clone of MOO)

<FOO>XY<BAR>ZW</BAR>Q</FOO>  -->  Y<BAR’>Z</BAR’>   (BAR’ is a clone of BAR)
```

It is important to note that nodes which are only partially contained by the range are cloned. Since part of such a node’s contents must remain in the original document (or document fragment) and part of the contents must be moved to the new fragment, a clone of the partially contained node is brought along to the new fragment. Note that cloning does not take place for "completely" contained elements - these elements are directly moved to the new fragment.

7.8. Cloning Content

The contents of a range may be duplicated using the following method:

```
DocumentFragment cloneContents ( );
```

This method returns a document fragment that is similar to the one returned by the method extractContents. However, in this case, the original nodes and text content in the range are not deleted from the original document. Instead, all of the nodes and text content within the returned document fragment are cloned.

7.9. Inserting Content

A node may be inserted into a range using the following method:

```
void insertNode ( in Node n );
```

The insertNode method inserts the specified node into the document or document fragment in which the range resides. For this method, the end position of the range is ignored and the node is inserted at the start position of the range.

The Node passed into this method can be a DocumentFragment. In that case, the contents of the fragment are inserted at the start position of the range, but the fragment itself is not. Note that if the Node represents the root of a sub-tree, the entire sub-tree is inserted.

Note that the same rules that apply to the insertBefore method on the Node interface apply here. Specifically, the Node passed in will be removed from its existing position in the same document or another fragment.

7.10. Surrounding Content

The insertion of a single element to subsume the content selected by range can be performed with:

```
void surroundContents ( in Node n );
```

The surroundContents member differs from insertNode in that surroundContents causes all of the content selected by the range to become children of the node argument, while insertNode splices in existing content at the given point in the document.

For example,calling surround contents with the node FOO yields:

```
Before:
  <BAR>AB<MOO>C</MOO>DE</BAR>
After surroundContents ( FOO ):
  <BAR>A<FOO>B<MOO>C</MOO>D</FOO>E</BAR>
```

Effectively, the surroundContents member modifies the document such that the begin tag of the node argument to be placed at the beginning of the range, and the end tag of the node argument to be placed at the end of the range. Of course, tags are not really being manipulated, however the effect is the same thus giving meaning to this member’s name: surroundContents.

Another way of of describing the effect of this member is to decompose it in terms of other operations:

1. Remove the contents selected by the range with a call to extractContents, saving away the selected contents into a new document fragment.
2. Insert the node argument where the range is now collapsed (after the extraction) with insertNode
3. Insert the entire contents of the extracted contents under the node argument.
4. Select the node argument and all of its contents with selectNode.

Because inserting a node in such a manor will be a common operation, surroundContents is provided to avoid the overhead of these four steps.

The surroundContents method may not be invoked in cases where the range only partially contains a non-Text node. Specifically, if the first non-Text node ancestor of the two end-points of a range is different, surroundContents will fail. An example of a range for which surroundContents may not be invoked is:

```
<FOO>AB<BAR>CD</BAR>E</FOO>
```

If the node argument has any children, those children are removed before its insertion. Also, if the node argument is part of any existing content, it is also removed from that content before insertion.

7.11. Miscellaneous Members

One can clone a range:

```
Range cloneRange ( );
```

This creates a new range which selects exactly the same content of the range on which it was called. No content is affected by this operation.

Because the end-points of a range do not have to necessarily share the same parent nodes, use:

```
readonly attribute Node commonParent;
```

to get the first node which is common to both endpoints. This is accomplished by walking up the parent chain of the two endpoints, locating the first node which is common.

One can get a copy of all the text nodes (or partial text nodes) selected by a range with:

```
domstring toString ( );
```

This does nothing more than simply concatenate all the textual content subsumed by the range.

7.12. Range behavior under document mutation

As the document is mutated, the Ranges within the document need to be updated. For example, if both ends of a Range are within the same 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, loosely speaking, all Ranges will select the same portion of the document after any mutation operation, where that is possible.

Any mutation of the document tree which affect Ranges can be considered to be a combination of basic delete and insertion operations. In fact, it can be convenient to think of those operations as being accomplished using the deleteContents() and insertNode() Range methods.
(**ED:** I think we also have to think of merging of TextNodes as a separate operation. Although the merge can be considered to be a deletion followed by an insertion, a Range which selected a portion of the text in the nodes being deleted won't select the same content after the merge. I think it should.)

7.12.1. Insertions

An insertion occurs at a single point in the document. Again, it is convenient to think of that point, called the insertion point, as the end-point of a Range. For any other Range in the document tree, consider each end-point. The only case in which the end-point will be changed after the insertion is when the end-point and the insertion point have the same parent Node and the offset of the insertion point is strictly less than the offset of the Range's end-point. In that case the offset of the Range's end-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 an end point, it is ambiguous as to where the end point should reposition itself if it wants to maintain its original relative position. It has two choices: either at the start or end of the newly inserted content. We have chosen to neither change the parent nor offset of the end-point in this case which means that it will be positioned at the start of the newly inserted content.

Examples:

In these examples, the portion of the document selected by the Range before and after the insertion will be shown as bold text.

Suppose the Range selects the following:

<P>Abcd efgh X**Y blah i**jkl</P>

Consider the insertion of the text "*inserted text*" in the following locations:

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>

Editor's NOTE:All of these results make intuitive sense except, perhaps, for example 2. where it might be expected that the result would be

<P>Abcd efgh X*inserted text***Y blah i**jkl</P>

7.12.2. Deletions

Any deletion from the document tree can be considered as a sequence of deleteContent() 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 deleteContent() operation of another Range. And, in fact, we need on consider what happens to a single end-point of the Range since both end-points will be modified using the same algorithm.

If an end-point is within the content being deleted, then it will be moved after the deletion to the same location as the common end-point of the Range used to delete the contents.

If an end-point is after the content being deleted then it is not affected by the deletion unless its parent node is also the parent node of one of the end-points of the range being deleted. If there is such a common parent, then the index of the end-point is modified so that the end-point maintains its position relative to the content of the parent.

If an end-point is before the content being deleted then it is not affect by the deletion at all.

Examples:

In these examples, the portion of the document selected by the Range before and after the insertion will be shown as bold text and the content being deleted is underlined. When the Range after the deletion is an insertion point, it will be shown as '^'.

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>
```

Example 4.

Before:

```
<P>Abcd efgh The Range ijkl</P>
```

After:

```
<P>Abcd he Range ijkl</P>
```

Example 5.

Before:

```
<P>Abcd <EM>efgh The Range ij</EM>kl</P>
```

After:

```
<P>Abcd ^kl</P>
```

7.13. Formal Description of the Range Interface

To summarize, here is the complete, formal description of the `Range` [p.74] interface:

Interface *Range*

IDL Definition

```
interface Range {
  readonly attribute  Node                 startParent;
  readonly attribute  long                 startOffset;
  readonly attribute  Node                 endParent;
  readonly attribute  long                 endOffset;
  readonly attribute  boolean              isCollapsed;
  readonly attribute  Node                 commonParent;
  void                      setStart(in Node parent, 
                                     in long offset)
                                     raises(RangeException);
  void                      setEnd(in Node parent, 
                                   in long offset)
                                   raises(RangeException);
  void                      setStartBefore(in Node sibling)
                                           raises(RangeException);
  void                      setStartAfter(in Node sibling)
                                          raises(RangeException);
  void                      setEndBefore(in Node sibling)
                                         raises(RangeException);
  void                      setEndAfter(in Node sibling)
                                        raises(RangeException);
  void                      collapse(in boolean toStart);
  void                      selectNode(in Node n)
                                       raises(RangeException);
  void                      selectNodeContents(in Node n)
                                               raises(RangeException);
  typedef enum CompareHow_ {
    StartToStart,
    StartToEnd,
    EndToEnd,
    EndToStart
  } CompareHow;
  short                     compareEndPoints(in CompareHow how,
                                             in Range sourceRange)
                                             raises(DOMException);
  void                      deleteContents()
                                           raises(DOMException);
  DocumentFragment          extractContents()
                                            raises(DOMException);
  DocumentFragment          cloneContents();
  void                      insertNode(in Node n)
                                       raises(DOMException, RangeException);
  void                      surroundContents(in Node n)
                                             raises(DOMException, RangeException);
  Range                     cloneRange();
  DOMString                 toString();
};
```

Attributes

`startParent`
: Node within which the range begins

`startOffset`
: Offset in the starting node of the start of the range.

`endParent`
: Node within which the range ends

`endOffset`
: Offset in the ending node of the end of the range.

`isCollapsed`
: TRUE if the range is collapsed

`commonParent`
: The common ancestor node of the entire range

Methods

`setStart`
: Sets the attribute values describing the start of the range.

Parameters

`parent`	The `startNode` value. This parameter must be non-null.
`offset`	The `startOffset` value.

Exceptions

`RangeException` [p.80]

NULL_PARENT_ERR: Raised if startNode is null.

INVALID_NODE_TYPE_ERR: Raised if an ancestor of startNode is an Attr, Entity, Notation or DocumentType node.

This method returns nothing.

`setEnd`

Sets the attributes describing the end of a range.

Parameters

`parent`	The `endNode` value. This parameter must be non-null.
`offset`	The `endOffset` value.

Exceptions

`RangeException` [p.80]

NULL_PARENT_ERR: Raised if endNode is null.

INVALID_NODE_TYPE_ERR: Raised if an ancestor of startNode is an Attr, Entity, Notation or DocumentType node.

This method returns nothing.

`setStartBefore`

Sets the starting position before a node

Parameters

`sibling`	Range starts before this node

Exceptions

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if an ancestor of sibling is an Attr, Entity, Notation or DocumentType node or if sibling itself is a Document or DocumentFragment node.

This method returns nothing.

`setStartAfter`

Sets the starting position after a node

Parameters

`sibling`	Range starts after this node

Exceptions

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if an ancestor of sibling is an Attr, Entity, Notation or DocumentType node or if sibling itself is a Document or DocumentFragment node.

This method returns nothing.

`setEndBefore`

Sets the ending position of a range to be before a given node.

Parameters

`sibling`	Range ends before this node

Exceptions

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if an ancestor of sibling is an Attr, Entity, Notation or DocumentType node or if sibling itself is a Document or DocumentFragment node.

This method returns nothing.

`setEndAfter`

Sets the ending position of a range to be after a given node

Parameters

`sibling`	Range ends after this node.

Exceptions

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if an ancestor of sibling is an Attr, Entity, Notation or DocumentType node or if sibling itself is a Document or DocumentFragment node.

This method returns nothing.

`collapse`

Collapse a range onto one of the end points

Parameters

`toStart`	If TRUE, collapses onto the starting node; if FALSE, collapses the range onto the ending node.

This method returns nothing.

This method raises no exceptions.

`selectNode`

Select a node and its contents

Parameters

`n`	Node to select from

Exceptions

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if an ancestor of n is an Attr, Entity, Notation or DocumentType node or if n itself is a Document or DocumentFragment node.

This method returns nothing.

`selectNodeContents`

Select the contents within a node

Parameters

`n`	Node to select from

Exceptions

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if an ancestor of n is an Attr, Entity, Notation or DocumentType node.

This method returns nothing.

Type Definition *CompareHow*

Enumeration *CompareHow_*

Enumerator Values

StartToStart	
StartToEnd	
EndToEnd	
EndToStart	

Methods

`compareEndPoints`

Compare the end-points of two ranges in a document.

Parameters

`how`

`sourceRange`

Return Value

-1, 0 or 1 depending on whether the corresponding end-point of the Range is less than, equal or greater than the corresponding end-point of sourceRange.

Exceptions

`DOMException`

WRONG_DOCUMENT_ERR: Raised if the two Ranges are not in the same document or document fragment.

`deleteContents`

Removes the contents of a range from the containing document or document fragment without returning a reference to the removed content.

Exceptions

`DOMException`

NO_MODIFICATION_ALLOWED_ERR: Raised if any portion of the content of the range is readonly or any of the nodes which contain any of the content of the range are readonly.

This method has no parameters.

This method returns nothing.

`extractContents`

Moves the contents of a range from the containing document or document fragment to a new DocumentFragment.

Return Value

A DocumentFragment containing the extracted contents.

Exceptions

`DOMException`

NO_MODIFICATION_ALLOWED_ERR: Raised if any portion of the content of the range is readonly or any of the nodes which contain any of the content of the range are readonly.

This method has no parameters.

`cloneContents`

Duplicates the contents of a range

Return Value

A DocumentFragment containing contents equivalent to those of this range.

This method has no parameters.

This method raises no exceptions.

`insertNode`

inserts the specified node into the document or document fragment at the start end-point of the range.

Parameters

`n`	The node to insert at the start end-point of the range

Exceptions

`DOMException`

NO_MODIFICATION_ALLOWED_ERR: Raised if the parent or any ancestor of the start end-point of the range is readonly.

`RangeException` [p.80]

INVALID_NODE_TYPE_ERR: Raised if n is an Attr, Entity, Notation, DocumentType or Document node.

This method returns nothing.

`surroundContents`

Reparents the contents of the range to the given node and inserts the node at the location of the start end-point of the range.

Parameters

`n`	The node to surround the contents with.

Exceptions

`DOMException`

NO_MODIFICATION_ALLOWED_ERR: Raised if the parent or any ancestor of the either end-point of the range is readonly.

`RangeException` [p.80]

BAD_ENDPOINTS_ERR: Raised if the range only partially contains a node.

INVALID_NODE_TYPE_ERR: Raised if n is an Attr, Entity, DocumentType, Notation, Document or DocumentFragment node.

This method returns nothing.

`cloneRange`

Produces a new range whose end-points are equal to the end-points of the range.

Return Value

The duplicated range.

This method has no parameters.

This method raises no exceptions.

`toString`

Returns the contents of a range as a string.

Return Value

The contents of the range.

This method has no parameters.

This method raises no exceptions.

Exception *RangeException*

The Range object needs additional exception codes to those in DOM Level 1. These codes will need to be consolidated with other exceptions added to DOM Level 2.

IDL Definition

```
exception RangeException {
  unsigned short   code;
};

// RangeExceptionCode
const unsigned short      BAD_ENDPOINTS_ERR  = 201;
const unsigned short      INVALID_NODE_TYPE_ERR = 202;
const unsigned short      NULL_PARENT_ERR    = 203;
```

Definition group *RangeExceptionCode*

An integer indicating the type of error generated.
Defined Constants

BAD_ENDPOINTS_ERR	If the end-points of a range do not meet specific requirements.
INVALID_NODE_TYPE_ERR	If the parent of an end-point of a range is being set using either a node with an ancestor of an invalid type or a node with an invalid type.
NULL_PARENT_ERR	If the parent of an end-point of a range is being set to null.

Appendix A: Contributors

Members of the DOM Working Group and Interest Group contributing to this specification were:

Lauren Wood, SoftQuad Software Inc., *chair*
Arnaud Le Hors, W3C, *W3C staff contact*
Andy Heninger, IBM
Bill Smith, Sun
Bill Shea, Merrill Lynch
Bob Sutor, IBM
Chris Wilson, Microsoft
David Brownell, Sun
Don Park, Docuverse
Eric Vasilik, Microsoft
Gavin Nicol, INSO
Jared Sorensen, Novell
Joe Kesselman, IBM
Joe Lapp, webMethods
Jonathan Robie, Texcel
Mike Champion, Arbortext
Peter Sharpe, SoftQuad Software Inc.
Ramesh Lekshmynarayanan, Merrill Lynch
Ray Whitmer, iMall
Rich Rollman, Microsoft
Tom Pixley, Netscape
Vidur Apparao, Netscape

Appendix A: Contributors

Appendix B: Glossary

Editors
Robert S. Sutor, IBM Research

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

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

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

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

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

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

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

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

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

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

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

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

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

cursor

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

data model

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

deprecation

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

descendant

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

ECMAScript

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

element

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

event propagation, also known as event bubbling

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

equivalence

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

hosting implementation

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

HTML

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

uncooked model
: See initial structure model.

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

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

Appendix B: Glossary

Appendix C: IDL Definitions

This appendix contains the complete OMG IDL for the Level 1 Document Object Model definitions. The definitions are divided into Stylesheets [p.91] , CSS [p.91] , Events [p.95] , Filters and Iterators [p.96] , and Range [p.97] .

C.1: Document Object Model Level 2 Stylesheets

stylesheets.idl:

```
// File: stylesheets.idl
#ifndef _STYLESHEETS_IDL_
#define _STYLESHEETS_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module stylesheets
{
  typedef dom::DOMString DOMString;
  typedef dom::Node Node;

  interface StyleSheet {
    readonly attribute  DOMString            type;
             attribute  boolean              disabled;
    readonly attribute  Node                 owningNode;
    readonly attribute  StyleSheet           parentStyleSheet;
    readonly attribute  DOMString            href;
    readonly attribute  DOMString            title;
    readonly attribute  DOMString            media;
  };

  interface StyleSheetCollection {
    readonly attribute  unsigned long        length;
    StyleSheet                item(in unsigned long index);
  };

};

#endif // _STYLESHEETS_IDL_
```

C.2: Document Object Model Level 2 CSS

css.idl:

```
// File: css.idl
#ifndef _CSS_IDL_
#define _CSS_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
```

```
module css
{
  typedef dom::DOMString DOMString;
  typedef dom::StyleSheet StyleSheet;

  interface CSSRule;
  interface CSSStyleSheet;
  interface CSSStyleDeclaration;

  interface CSSRuleCollection {
    readonly attribute  unsigned long        length;
    CSSRule                    item(in unsigned long index);
  };

  interface CSSRule {
    // RuleType
    const unsigned short      UNKNOWN_RULE       = 0;
    const unsigned short      STYLE_RULE         = 1;
    const unsigned short      IMPORT_RULE        = 2;
    const unsigned short      MEDIA_RULE         = 3;
    const unsigned short      FONT_FACE_RULE     = 4;
    const unsigned short      PAGE_RULE          = 5;

    readonly attribute  unsigned short       type;
             attribute  DOMString            cssText;
                                 // raises(DOMException) on setting
    readonly attribute  CSSStyleSheet        parentStyleSheet;
    readonly attribute  CSSRule              parentRule;
  };

  interface CSSStyleRule : CSSRule {
             attribute  DOMString            selectorText;
    readonly attribute  CSSStyleDeclaration  style;
  };

  interface CSSMediaRule : CSSRule {
             attribute  DOMString            mediaTypes;
    readonly attribute  CSSRuleCollection    cssRules;
    unsigned long              insertRule(in DOMString rule,
                                          in unsigned long index)
                                          raises(DOMException);
    void                       deleteRule(in unsigned long index);
  };

  interface CSSFontFaceRule : CSSRule {
    readonly attribute  CSSStyleDeclaration  style;
  };

  interface CSSPageRule : CSSRule {
             attribute  DOMString            selectorText;
    readonly attribute  CSSStyleDeclaration  style;
  };

  interface CSSImportRule : CSSRule {
             attribute  DOMString            href;
             attribute  DOMString            media;
    readonly attribute  CSSStyleSheet        styleSheet;
```

```
};

interface CSSUnknownRule : CSSRule {
};

interface CSSStyleDeclaration {
           attribute  DOMString            cssText;
                                       // raises(DOMException) on setting
  DOMString                 getPropertyValue(in DOMString propertyName);
  DOMString                 removeProperty(in DOMString propertyName);
  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;
};

interface CSS2Properties {
           attribute  DOMString            azimuth;
           attribute  DOMString            background;
           attribute  DOMString            backgroundAttachment;
           attribute  DOMString            backgroundColor;
           attribute  DOMString            backgroundImage;
           attribute  DOMString            backgroundPosition;
           attribute  DOMString            backgroundRepeat;
           attribute  DOMString            border;
           attribute  DOMString            borderCollapse;
           attribute  DOMString            borderColor;
           attribute  DOMString            borderSpacing;
           attribute  DOMString            borderStyle;
           attribute  DOMString            borderTop;
           attribute  DOMString            borderRight;
           attribute  DOMString            borderBottom;
           attribute  DOMString            borderLeft;
           attribute  DOMString            borderTopColor;
           attribute  DOMString            borderRightColor;
           attribute  DOMString            borderBottomColor;
           attribute  DOMString            borderLeftColor;
           attribute  DOMString            borderTopStyle;
           attribute  DOMString            borderRightStyle;
           attribute  DOMString            borderBottomStyle;
           attribute  DOMString            borderLeftStyle;
           attribute  DOMString            borderTopWidth;
           attribute  DOMString            borderRightWidth;
           attribute  DOMString            borderBottomWidth;
           attribute  DOMString            borderLeftWidth;
           attribute  DOMString            borderWidth;
           attribute  DOMString            bottom;
           attribute  DOMString            captionSide;
           attribute  DOMString            clear;
           attribute  DOMString            clip;
           attribute  DOMString            color;
           attribute  DOMString            content;
           attribute  DOMString            counterIncrement;
           attribute  DOMString            counterReset;
           attribute  DOMString            cue;
           attribute  DOMString            cueAfter;
           attribute  DOMString            cueBefore;
           attribute  DOMString            cursor;
           attribute  DOMString            direction;
           attribute  DOMString            display;
           attribute  DOMString            elevation;
           attribute  DOMString            emptyCells;
           attribute  DOMString            cssFloat;
           attribute  DOMString            font;
           attribute  DOMString            fontFamily;
           attribute  DOMString            fontSize;
           attribute  DOMString            fontSizeAdjust;
           attribute  DOMString            fontStretch;
           attribute  DOMString            fontStyle;
           attribute  DOMString            fontVariant;
           attribute  DOMString            fontWeight;
           attribute  DOMString            height;
           attribute  DOMString            left;
           attribute  DOMString            letterSpacing;
           attribute  DOMString            lineHeight;
           attribute  DOMString            listStyle;
           attribute  DOMString            listStyleImage;
           attribute  DOMString            listStylePosition;
           attribute  DOMString            listStyleType;
           attribute  DOMString            margin;
           attribute  DOMString            marginTop;
           attribute  DOMString            marginRight;
           attribute  DOMString            marginBottom;
           attribute  DOMString            marginLeft;
           attribute  DOMString            markerOffset;
           attribute  DOMString            marks;
           attribute  DOMString            maxHeight;
           attribute  DOMString            maxWidth;
           attribute  DOMString            minHeight;
           attribute  DOMString            minWidth;
           attribute  DOMString            orphans;
           attribute  DOMString            outline;
           attribute  DOMString            outlineColor;
           attribute  DOMString            outlineStyle;
           attribute  DOMString            outlineWidth;
           attribute  DOMString            overflow;
           attribute  DOMString            padding;
           attribute  DOMString            paddingTop;
           attribute  DOMString            paddingRight;
           attribute  DOMString            paddingBottom;
           attribute  DOMString            paddingLeft;
           attribute  DOMString            page;
           attribute  DOMString            pageBreakAfter;
           attribute  DOMString            pageBreakBefore;
           attribute  DOMString            pageBreakInside;
           attribute  DOMString            pause;
           attribute  DOMString            pauseAfter;
           attribute  DOMString            pauseBefore;
           attribute  DOMString            pitch;
           attribute  DOMString            pitchRange;
```

```
             attribute  DOMString            playDuring;
             attribute  DOMString            position;
             attribute  DOMString            quotes;
             attribute  DOMString            richness;
             attribute  DOMString            right;
             attribute  DOMString            size;
             attribute  DOMString            speak;
             attribute  DOMString            speakHeader;
             attribute  DOMString            speakNumeral;
             attribute  DOMString            speakPunctuation;
             attribute  DOMString            speechRate;
             attribute  DOMString            stress;
             attribute  DOMString            tableLayout;
             attribute  DOMString            textAlign;
             attribute  DOMString            textDecoration;
             attribute  DOMString            textIndent;
             attribute  DOMString            textShadow;
             attribute  DOMString            textTransform;
             attribute  DOMString            top;
             attribute  DOMString            unicodeBidi;
             attribute  DOMString            verticalAlign;
             attribute  DOMString            visibility;
             attribute  DOMString            voiceFamily;
             attribute  DOMString            volume;
             attribute  DOMString            whiteSpace;
             attribute  DOMString            widows;
             attribute  DOMString            width;
             attribute  DOMString            wordSpacing;
             attribute  DOMString            zIndex;
  };

  interface CSSStyleSheet : StyleSheet {
    readonly attribute  CSSRuleCollection    cssRules;
    unsigned long             insertRule(in DOMString rule, 
                                         in unsigned long index)
                                         raises(DOMException);
    void                      deleteRule(in unsigned long index)
                                         raises(DOMException);
  };

};

#endif // _CSS_IDL_
```

C.3: Document Object Model Level 2 Events

events.idl:

```
// File: events.idl
#ifndef _EVENTS_IDL_
#define _EVENTS_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
```

```
module events
{
  typedef dom::DOMString DOMString;
  typedef dom::Node Node;

  interface EventListener;
  interface Event;

  interface EventTarget {
    void                      addEventListener(in DOMString type,
                                               in boolean postProcess,
                                               in boolean useCapture,
                                               in EventListener listener);
    void                      removeEventListener(in DOMString type,
                                                  in boolean postProcess,
                                                  in boolean useCapture,
                                                  in EventListener listener);
  };

  interface EventListener {
    void                      handleEvent(in Event event);
  };

  interface Event {
             attribute  DOMString            type;
             attribute  Node                 target;
             attribute  Node                 currentNode;
             attribute  boolean              cancelBubble;
             attribute  boolean              cancelCapture;
             attribute  boolean              returnValue;
  };

  interface UIEvent : Event {
             attribute  long                 screenX;
             attribute  long                 screenY;
             attribute  long                 clientX;
             attribute  long                 clientY;
             attribute  boolean              altKey;
             attribute  boolean              ctrlKey;
             attribute  boolean              shiftKey;
             attribute  unsigned long        keyCode;
             attribute  unsigned long        charCode;
             attribute  unsigned short       button;
  };

  interface MutationEvent : Event {
             attribute  Node                 relatedNode;
             attribute  DOMString            prevValue;
             attribute  DOMString            newValue;
             attribute  DOMString            attrName;
  };

};

#endif // _EVENTS_IDL_
```

C.4: Document Object Model Level 2 Filters and Iterators

fi.idl:

```
// File: fi.idl
#ifndef _FI_IDL_
#define _FI_IDL_

#include "dom.idl"

#pragma prefix "dom.w3c.org"
module fi
{
  typedef dom::Node Node;

  interface NodeIterator {
    Node                      nextNode();
    Node                      prevNode();
  };

  interface Document {
    boolean                   createTreeIterator(in Node root, 
                                                 in short whatToShow);
  };

  interface NodeFilter {
    boolean                   acceptNode(in Node n);
  };

};

#endif // _FI_IDL_
```

C.5: Document Object Model Level 2 Range

range.idl:

```
// 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;

  exception RangeException {
    unsigned short   code;
  };
  // RangeExceptionCode
  const unsigned short      BAD_ENDPOINTS_ERR  = 201;
  const unsigned short      INVALID_NODE_TYPE_ERR = 202;
  const unsigned short      NULL_PARENT_ERR    = 203;


  interface Range {
    readonly attribute  Node                 startParent;
    readonly attribute  long                 startOffset;
    readonly attribute  Node                 endParent;
    readonly attribute  long                 endOffset;
    readonly attribute  boolean              isCollapsed;
    readonly attribute  Node                 commonParent;
    void                      setStart(in Node parent,
                                       in long offset)
                                       raises(RangeException);
    void                      setEnd(in Node parent,
                                     in long offset)
                                     raises(RangeException);
    void                      setStartBefore(in Node sibling)
                                             raises(RangeException);
    void                      setStartAfter(in Node sibling)
                                            raises(RangeException);
    void                      setEndBefore(in Node sibling)
                                           raises(RangeException);
    void                      setEndAfter(in Node sibling)
                                          raises(RangeException);
    void                      collapse(in boolean toStart);
    void                      selectNode(in Node n)
                                         raises(RangeException);
    void                      selectNodeContents(in Node n)
                                                 raises(RangeException);
    typedef enum CompareHow_ {
      StartToStart,
      StartToEnd,
      EndToEnd,
      EndToStart
    } CompareHow;
    short                     compareEndPoints(in CompareHow how,
                                               in Range sourceRange)
                                               raises(DOMException);
    void                      deleteContents()
                                             raises(DOMException);
    DocumentFragment          extractContents()
                                              raises(DOMException);
    DocumentFragment          cloneContents();
    void                      insertNode(in Node n)
                                         raises(DOMException, RangeException);
    void                      surroundContents(in Node n)
                                               raises(DOMException, RangeException);
    Range                     cloneRange();
    DOMString                 toString();
  };

};

#endif // _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 Stylesheets [p.99] , CSS [p.99] , Events [p.107] , Filters and Iterators [p.108] , and Range [p.109] .

D.1: Document Object Model Level 2 Stylesheets

org/w3c/dom/stylesheets/StyleSheet.java:

```
package org.w3c.dom.stylesheets;

import org.w3c.dom.*;

public interface StyleSheet {
  public String             getType();
  public boolean            getDisabled();
  public void               setDisabled(boolean disabled);
  public Node               getOwningNode();
  public StyleSheet         getParentStyleSheet();
  public String             getHref();
  public String             getTitle();
  public String             getMedia();
}
```

org/w3c/dom/stylesheets/StyleSheetCollection.java:

```
package org.w3c.dom.stylesheets;

import org.w3c.dom.*;

public interface StyleSheetCollection {
  public int                getLength();
  public StyleSheet         item(int index);
}
```

D.2: Document Object Model Level 2 CSS

org/w3c/dom/css/CSSStyleSheet.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSStyleSheet extends StyleSheet {
  public CSSRuleCollection  getCssRules();
  public int                insertRule(String rule,
                                       int index)
                                       throws DOMException;
  public void                deleteRule(int index)
                                       throws DOMException;
}
```

org/w3c/dom/css/CSSRuleCollection.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSRuleCollection {
  public int                 getLength();
  public CSSRule             item(int index);
}
```

org/w3c/dom/css/CSSRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSRule {
  // RuleType
  public static final short           UNKNOWN_RULE         = 0;
  public static final short           STYLE_RULE           = 1;
  public static final short           IMPORT_RULE          = 2;
  public static final short           MEDIA_RULE           = 3;
  public static final short           FONT_FACE_RULE       = 4;
  public static final short           PAGE_RULE            = 5;

  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.*;

public interface CSSStyleRule extends CSSRule {
  public String              getSelectorText();
  public void                setSelectorText(String selectorText);
  public CSSStyleDeclaration getStyle();
}
```

org/w3c/dom/css/CSSMediaRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSMediaRule extends CSSRule {
  public String             getMediaTypes();
  public void               setMediaTypes(String mediaTypes);
  public CSSRuleCollection  getCssRules();
  public int                insertRule(String rule,
                                       int index)
                                       throws DOMException;
  public void               deleteRule(int index);
}
```

org/w3c/dom/css/CSSFontFaceRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSFontFaceRule extends CSSRule {
  public CSSStyleDeclaration getStyle();
}
```

org/w3c/dom/css/CSSPageRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSPageRule extends CSSRule {
  public String             getSelectorText();
  public void               setSelectorText(String selectorText);
  public CSSStyleDeclaration getStyle();
}
```

org/w3c/dom/css/CSSImportRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSImportRule extends CSSRule {
  public String             getHref();
  public void               setHref(String href);
  public String             getMedia();
  public void               setMedia(String media);
  public CSSStyleSheet      getStyleSheet();
}
```

org/w3c/dom/css/CSSUnknownRule.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSUnknownRule extends CSSRule {
}
```

org/w3c/dom/css/CSSStyleDeclaration.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSSStyleDeclaration {
  public String             getCssText();
  public void               setCssText(String cssText)
                                       throws DOMException;
  public String             getPropertyValue(String propertyName);
  public String             removeProperty(String propertyName);
  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/CSS2Properties.java:

```
package org.w3c.dom.css;

import org.w3c.dom.*;

public interface CSS2Properties {
  public String             getAzimuth();
  public void               setAzimuth(String azimuth);
  public String             getBackground();
  public void               setBackground(String background);
  public String             getBackgroundAttachment();
  public void               setBackgroundAttachment(String backgroundAttachment);
  public String             getBackgroundColor();
  public void               setBackgroundColor(String backgroundColor);
  public String             getBackgroundImage();
  public void               setBackgroundImage(String backgroundImage);
  public String             getBackgroundPosition();
  public void               setBackgroundPosition(String backgroundPosition);
  public String             getBackgroundRepeat();
  public void               setBackgroundRepeat(String backgroundRepeat);
  public String             getBorder();
  public void               setBorder(String border);
  public String             getBorderCollapse();
  public void             setBorderCollapse(String borderCollapse);
  public String           getBorderColor();
  public void             setBorderColor(String borderColor);
  public String           getBorderSpacing();
  public void             setBorderSpacing(String borderSpacing);
  public String           getBorderStyle();
  public void             setBorderStyle(String borderStyle);
  public String           getBorderTop();
  public void             setBorderTop(String borderTop);
  public String           getBorderRight();
  public void             setBorderRight(String borderRight);
  public String           getBorderBottom();
  public void             setBorderBottom(String borderBottom);
  public String           getBorderLeft();
  public void             setBorderLeft(String borderLeft);
  public String           getBorderTopColor();
  public void             setBorderTopColor(String borderTopColor);
  public String           getBorderRightColor();
  public void             setBorderRightColor(String borderRightColor);
  public String           getBorderBottomColor();
  public void             setBorderBottomColor(String borderBottomColor);
  public String           getBorderLeftColor();
  public void             setBorderLeftColor(String borderLeftColor);
  public String           getBorderTopStyle();
  public void             setBorderTopStyle(String borderTopStyle);
  public String           getBorderRightStyle();
  public void             setBorderRightStyle(String borderRightStyle);
  public String           getBorderBottomStyle();
  public void             setBorderBottomStyle(String borderBottomStyle);
  public String           getBorderLeftStyle();
  public void             setBorderLeftStyle(String borderLeftStyle);
  public String           getBorderTopWidth();
  public void             setBorderTopWidth(String borderTopWidth);
  public String           getBorderRightWidth();
  public void             setBorderRightWidth(String borderRightWidth);
  public String           getBorderBottomWidth();
  public void             setBorderBottomWidth(String borderBottomWidth);
  public String           getBorderLeftWidth();
  public void             setBorderLeftWidth(String borderLeftWidth);
  public String           getBorderWidth();
  public void             setBorderWidth(String borderWidth);
  public String           getBottom();
  public void             setBottom(String bottom);
  public String           getCaptionSide();
  public void             setCaptionSide(String captionSide);
  public String           getClear();
  public void             setClear(String clear);
  public String           getClip();
  public void             setClip(String clip);
  public String           getColor();
  public void             setColor(String color);
  public String           getContent();
  public void             setContent(String content);
  public String           getCounterIncrement();
  public void             setCounterIncrement(String counterIncrement);
  public String           getCounterReset();
  public void             setCounterReset(String counterReset);
  public String             getCue();
  public void               setCue(String cue);
  public String             getCueAfter();
  public void               setCueAfter(String cueAfter);
  public String             getCueBefore();
  public void               setCueBefore(String cueBefore);
  public String             getCursor();
  public void               setCursor(String cursor);
  public String             getDirection();
  public void               setDirection(String direction);
  public String             getDisplay();
  public void               setDisplay(String display);
  public String             getElevation();
  public void               setElevation(String elevation);
  public String             getEmptyCells();
  public void               setEmptyCells(String emptyCells);
  public String             getCssFloat();
  public void               setCssFloat(String cssFloat);
  public String             getFont();
  public void               setFont(String font);
  public String             getFontFamily();
  public void               setFontFamily(String fontFamily);
  public String             getFontSize();
  public void               setFontSize(String fontSize);
  public String             getFontSizeAdjust();
  public void               setFontSizeAdjust(String fontSizeAdjust);
  public String             getFontStretch();
  public void               setFontStretch(String fontStretch);
  public String             getFontStyle();
  public void               setFontStyle(String fontStyle);
  public String             getFontVariant();
  public void               setFontVariant(String fontVariant);
  public String             getFontWeight();
  public void               setFontWeight(String fontWeight);
  public String             getHeight();
  public void               setHeight(String height);
  public String             getLeft();
  public void               setLeft(String left);
  public String             getLetterSpacing();
  public void               setLetterSpacing(String letterSpacing);
  public String             getLineHeight();
  public void               setLineHeight(String lineHeight);
  public String             getListStyle();
  public void               setListStyle(String listStyle);
  public String             getListStyleImage();
  public void               setListStyleImage(String listStyleImage);
  public String             getListStylePosition();
  public void               setListStylePosition(String listStylePosition);
  public String             getListStyleType();
  public void               setListStyleType(String listStyleType);
  public String             getMargin();
  public void               setMargin(String margin);
  public String             getMarginTop();
  public void               setMarginTop(String marginTop);
  public String             getMarginRight();
  public void               setMarginRight(String marginRight);
  public String             getMarginBottom();
  public void             setMarginBottom(String marginBottom);
  public String           getMarginLeft();
  public void             setMarginLeft(String marginLeft);
  public String           getMarkerOffset();
  public void             setMarkerOffset(String markerOffset);
  public String           getMarks();
  public void             setMarks(String marks);
  public String           getMaxHeight();
  public void             setMaxHeight(String maxHeight);
  public String           getMaxWidth();
  public void             setMaxWidth(String maxWidth);
  public String           getMinHeight();
  public void             setMinHeight(String minHeight);
  public String           getMinWidth();
  public void             setMinWidth(String minWidth);
  public String           getOrphans();
  public void             setOrphans(String orphans);
  public String           getOutline();
  public void             setOutline(String outline);
  public String           getOutlineColor();
  public void             setOutlineColor(String outlineColor);
  public String           getOutlineStyle();
  public void             setOutlineStyle(String outlineStyle);
  public String           getOutlineWidth();
  public void             setOutlineWidth(String outlineWidth);
  public String           getOverflow();
  public void             setOverflow(String overflow);
  public String           getPadding();
  public void             setPadding(String padding);
  public String           getPaddingTop();
  public void             setPaddingTop(String paddingTop);
  public String           getPaddingRight();
  public void             setPaddingRight(String paddingRight);
  public String           getPaddingBottom();
  public void             setPaddingBottom(String paddingBottom);
  public String           getPaddingLeft();
  public void             setPaddingLeft(String paddingLeft);
  public String           getPage();
  public void             setPage(String page);
  public String           getPageBreakAfter();
  public void             setPageBreakAfter(String pageBreakAfter);
  public String           getPageBreakBefore();
  public void             setPageBreakBefore(String pageBreakBefore);
  public String           getPageBreakInside();
  public void             setPageBreakInside(String pageBreakInside);
  public String           getPause();
  public void             setPause(String pause);
  public String           getPauseAfter();
  public void             setPauseAfter(String pauseAfter);
  public String           getPauseBefore();
  public void             setPauseBefore(String pauseBefore);
  public String           getPitch();
  public void             setPitch(String pitch);
  public String           getPitchRange();
  public void             setPitchRange(String pitchRange);
  public String           getPlayDuring();
  public void             setPlayDuring(String playDuring);
```

```
  public String             getPosition();
  public void               setPosition(String position);
  public String             getQuotes();
  public void               setQuotes(String quotes);
  public String             getRichness();
  public void               setRichness(String richness);
  public String             getRight();
  public void               setRight(String right);
  public String             getSize();
  public void               setSize(String size);
  public String             getSpeak();
  public void               setSpeak(String speak);
  public String             getSpeakHeader();
  public void               setSpeakHeader(String speakHeader);
  public String             getSpeakNumeral();
  public void               setSpeakNumeral(String speakNumeral);
  public String             getSpeakPunctuation();
  public void               setSpeakPunctuation(String speakPunctuation);
  public String             getSpeechRate();
  public void               setSpeechRate(String speechRate);
  public String             getStress();
  public void               setStress(String stress);
  public String             getTableLayout();
  public void               setTableLayout(String tableLayout);
  public String             getTextAlign();
  public void               setTextAlign(String textAlign);
  public String             getTextDecoration();
  public void               setTextDecoration(String textDecoration);
  public String             getTextIndent();
  public void               setTextIndent(String textIndent);
  public String             getTextShadow();
  public void               setTextShadow(String textShadow);
  public String             getTextTransform();
  public void               setTextTransform(String textTransform);
  public String             getTop();
  public void               setTop(String top);
  public String             getUnicodeBidi();
  public void               setUnicodeBidi(String unicodeBidi);
  public String             getVerticalAlign();
  public void               setVerticalAlign(String verticalAlign);
  public String             getVisibility();
  public void               setVisibility(String visibility);
  public String             getVoiceFamily();
  public void               setVoiceFamily(String voiceFamily);
  public String             getVolume();
  public void               setVolume(String volume);
  public String             getWhiteSpace();
  public void               setWhiteSpace(String whiteSpace);
  public String             getWidows();
  public void               setWidows(String widows);
  public String             getWidth();
  public void               setWidth(String width);
  public String             getWordSpacing();
  public void               setWordSpacing(String wordSpacing);
  public String             getZIndex();
  public void               setZIndex(String zIndex);
}
```

D.3: Document Object Model Level 2 Events

org/w3c/dom/events/EventTarget.java:

```
package org.w3c.dom.events;

import org.w3c.dom.*;

public interface EventTarget {
  public void               addEventListener(String type,
                                             boolean postProcess,
                                             boolean useCapture,
                                             EventListener listener);
  public void               removeEventListener(String type,
                                                boolean postProcess,
                                                boolean useCapture,
                                                EventListener listener);
}
```

org/w3c/dom/events/EventListener.java:

```
package org.w3c.dom.events;

import org.w3c.dom.*;

public interface EventListener {
  public void               handleEvent(Event event);
}
```

org/w3c/dom/events/Event.java:

```
package org.w3c.dom.events;

import org.w3c.dom.*;

public interface Event {
  public String             getType();
  public void               setType(String type);
  public Node               getTarget();
  public void               setTarget(Node target);
  public Node               getCurrentNode();
  public void               setCurrentNode(Node currentNode);
  public boolean            getCancelBubble();
  public void               setCancelBubble(boolean cancelBubble);
  public boolean            getCancelCapture();
  public void               setCancelCapture(boolean cancelCapture);
  public boolean            getReturnValue();
  public void               setReturnValue(boolean returnValue);
}
```

org/w3c/dom/events/UIEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.*;

public interface UIEvent extends Event {
  public int               getScreenX();
  public void              setScreenX(int screenX);
  public int               getScreenY();
  public void              setScreenY(int screenY);
  public int               getClientX();
  public void              setClientX(int clientX);
  public int               getClientY();
  public void              setClientY(int clientY);
  public boolean           getAltKey();
  public void              setAltKey(boolean altKey);
  public boolean           getCtrlKey();
  public void              setCtrlKey(boolean ctrlKey);
  public boolean           getShiftKey();
  public void              setShiftKey(boolean shiftKey);
  public int               getKeyCode();
  public void              setKeyCode(int keyCode);
  public int               getCharCode();
  public void              setCharCode(int charCode);
  public short             getButton();
  public void              setButton(short button);
}
```

org/w3c/dom/events/MutationEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.*;

public interface MutationEvent extends Event {
  public Node              getRelatedNode();
  public void              setRelatedNode(Node relatedNode);
  public String            getPrevValue();
  public void              setPrevValue(String prevValue);
  public String            getNewValue();
  public void              setNewValue(String newValue);
  public String            getAttrName();
  public void              setAttrName(String attrName);
}
```

D.4: Document Object Model Level 2 Filters and Iterators

org/w3c/dom/fi/NodeIterator.java:

```
package org.w3c.dom.fi;

import org.w3c.dom.*;

public interface NodeIterator {
  public Node              nextNode();
  public Node              prevNode();
}
```

org/w3c/dom/fi/Document.java:

```
package org.w3c.dom.fi;

import org.w3c.dom.*;

public interface Document {
  public boolean           createTreeIterator(Node root,
                                              short whatToShow);
}
```

org/w3c/dom/fi/NodeFilter.java:

```
package org.w3c.dom.fi;

import org.w3c.dom.*;

public interface NodeFilter {
  public boolean           acceptNode(Node n);
}
```

D.5: Document Object Model Level 2 Range

org/w3c/dom/range/RangeException.java:

```
package org.w3c.dom.range;

import org.w3c.dom.*;

public abstract class RangeException extends RuntimeException {
  public RangeException(short code, String message) {
     super(message);
     this.code = code;
  }
  public short   code;
  // RangeExceptionCode
  public static final short           BAD_ENDPOINTS_ERR    = 201;
  public static final short           INVALID_NODE_TYPE_ERR = 202;
  public static final short           NULL_PARENT_ERR      = 203;

}
```

org/w3c/dom/range/Range.java:

```
package org.w3c.dom.range;

import org.w3c.dom.*;

public interface Range {
  public Node                getStartParent();
  public int                 getStartOffset();
  public Node                getEndParent();
  public int                 getEndOffset();
  public boolean             getIsCollapsed();
  public Node                getCommonParent();
  public void                setStart(Node parent,
                                      int offset)
                                      throws RangeException;
  public void                setEnd(Node parent,
                                    int offset)
                                    throws RangeException;
  public void                setStartBefore(Node sibling)
                                            throws RangeException;
  public void                setStartAfter(Node sibling)
                                           throws RangeException;
  public void                setEndBefore(Node sibling)
                                          throws RangeException;
  public void                setEndAfter(Node sibling)
                                         throws RangeException;
  public void                collapse(boolean toStart);
  public void                selectNode(Node n)
                                        throws RangeException;
  public void                selectNodeContents(Node n)
                                                throws RangeException;

  public static final int StartToStart = 1;
  public static final int StartToEnd   = 2;
  public static final int EndToEnd     = 3;
  public static final int EndToStart   = 4;


  public short               compareEndPoints(int how,
                                              Range sourceRange)
                                              throws DOMException;
  public void                deleteContents()
                                            throws DOMException;
  public DocumentFragment    extractContents()
                                             throws DOMException;
  public DocumentFragment    cloneContents();
  public void                insertNode(Node n)
                                        throws DOMException, RangeException;
  public void                surroundContents(Node n)
                                              throws DOMException, RangeException;
  public Range               cloneRange();
  public String              toString();
}
```

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 Stylesheets [p.111] , CSS [p.111] , Events [p.119] , Filters and Iterators [p.120] , and Range [p.121] .

E.1: Document Object Model Level 2 Stylesheets

Object **StyleSheet**

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

type

This property is of type **String**.

disabled

This property is of type **boolean**.

owningNode

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 **String**.

Object **StyleSheetCollection**

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

length

This property is of type **int**.

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

item(index)

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

E.2: Document Object Model Level 2 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:

cssRules

This property is of type **CSSRuleCollection**.

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 **CSSRuleCollection**
The **CSSRuleCollection** object has the following properties:
length
This property is of type **int**.
The **CSSRuleCollection** object has the following methods:
item(index)
This method returns a **CSSRule**. The **index** parameter is of type **unsigned long**.

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:
mediaTypes
This property is of type **String**.
cssRules
This property is of type **CSSRuleCollection**.
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 **String**.

styleSheet

This property is of type **CSSStyleSheet**.

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**.

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**.

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.3: Document Object Model Level 2 Events

Object **EventTarget**
The **EventTarget** object has the following methods:
addEventListener(type, postProcess, useCapture, listener)
This method returns a **void**. The **type** parameter is of type **DOMString**. The **postProcess** parameter is of type **boolean**. The **useCapture** parameter is of type **boolean**. The **listener** parameter is of type **EventListener**.
removeEventListener(type, postProcess, useCapture, listener)
This method returns a **void**. The **type** parameter is of type **DOMString**. The **postProcess** parameter is of type **boolean**. The **useCapture** parameter is of type **boolean**. The **listener** parameter is of type **EventListener**.
Object **EventListener**
The **EventListener** object has the following methods:
handleEvent(event)
This method returns a **void**. The **event** parameter is of type **Event**.
Object **Event**
The **Event** object has the following properties:
type
This property is of type **String**.
target
This property is of type **Node**.
currentNode
This property is of type **Node**.
cancelBubble
This property is of type **boolean**.
cancelCapture
This property is of type **boolean**.
returnValue
This property is of type **boolean**.

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:

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**.

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

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

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

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

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

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

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**.

E.4: Document Object Model Level 2 Filters and Iterators

Object **NodeIterator**

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

nextNode()
This method returns a **Node**.

prevNode()
This method returns a **Node**.

Object **Document**

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

createTreeIterator(root, whatToShow)

This method returns a **boolean**. The **root** parameter is of type **Node**. The **whatToShow** parameter is of type **short**.

Object **NodeFilter**

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

acceptNode(n)

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

E.5: Document Object Model Level 2 Range

Object **Range**

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

startParent

This property is of type **Node**.

startOffset

This property is of type **long**.

endParent

This property is of type **Node**.

endOffset

This property is of type **long**.

isCollapsed

This property is of type **boolean**.

commonParent

This property is of type **Node**.

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

setStart(parent, offset)

This method returns a **void**. The **parent** parameter is of type **Node**. The **offset** parameter is of type **long**.

setEnd(parent, offset)

This method returns a **void**. The **parent** parameter is of type **Node**. The **offset** parameter is of type **long**.

setStartBefore(sibling)

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

setStartAfter(sibling)

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

setEndBefore(sibling)

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

setEndAfter(sibling)

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

collapse(toStart)

This method returns a **void**. The **toStart** parameter is of type **boolean**.

selectNode(n)

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

selectNodeContents(n)
: This method returns a **void**. The **n** parameter is of type **Node**.

compareEndPoints(how, sourceRange)
: This method returns a **short**. The **how** parameter is of type **CompareHow**. 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(n)
: This method returns a **void**. The **n** parameter is of type **Node**.

surroundContents(n)
: This method returns a **void**. The **n** parameter is of type **Node**.

cloneRange()
: This method returns a **Range**.

toString()
: This method returns a **DOMString**.

References

CORBA

OMG (Object Management Group) *The Common Object Request Broker: Architecture and Specification*. See http://www.omg.org/corba/corbiiop.htm .

DOM-Level-1

W3C (World Wide Web Consortium) *DOM Level 1 Specification*. See http://www.w3.org/TR/REC-DOM-Level-1 .

ECMAScript

ECMA (European Computer Manufacturers Association) *ECMAScript Language Specification*. See http://www.ecma.ch/stand/ECMA-262.htm .

HTML4.0

W3C (World Wide Web Consortium) *HTML 4.0 Specification*. See http://www.w3.org/TR/REC-html40 .

Java

Sun *The Java Language Specification*. See http://java.sun.com/docs/books/jls/ .

Namespaces

W3C (World Wide Web Consortium) *Namespaces in XML* . See http://www.w3.org/TR/REC-xml-names .

Unicode

The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

XML

W3C (World Wide Web Consortium) *Extensible Markup Language (XML) 1.0*. See http://www.w3.org/TR/REC-xml .

Index