W3C DOM Reference

This part of the book is a reference section that documents the interfaces, methods, and properties defined by the W3C Level 1 and Level 2 DOM standards. Intermediate and advanced programmers who are writing for the newest generation of standards-compliant web browsers will use this reference section, in conjunction with the core and client-side JavaScript references in Parts III and IV. The introduction and sample reference page explain how to use and get the most out of this reference section. There are significant differences between this reference section and the other two, and you should read this introduction carefully so you can fully understand the reference information it contains.

Like the core and client-side references, this reference section is arranged alphabetically. The reference pages for the methods and properties of DOM interfaces are alphabetized by their full names, which include the names of the interfaces that define them. For example, if you want to read about the `appendChild()` method of the Node interface, you would look under "Node.appendChild," not just "appendChild."

To save space in this enlarged fourth edition of the book, properties in this reference section do not have reference pages of their own (all interfaces and methods do have their own reference pages, however). Instead, each property is completely documented in the reference page for the interface that defines it. For example, you can read about the `tagName` property of the Element interface in the "Element" reference page.

Sometimes you may find that you don't know the name of the interface that defines the method or property you want to look up, or you may not be sure which of the three reference sections to look up a class or interface in. Part VI of this book is a special index designed to help with these situations. Look up the name of a class, interface, method, or property, and it will tell you which reference section to look in and which class to look under in that section. For example, if you look up "Document," it will tell you that both the client-side and DOM reference sections have entries under that name. And if you look up the name "firstChild," it will tell you that `firstChild` is a property of Node, which you can read about in this DOM reference section.

Once you've found the reference page you're looking for, you shouldn't have much difficulty finding the information you need. Because the DOM standard is intended to work with languages other than JavaScript, however, it was written with typed languages (such as Java and C++) in mind. Although JavaScript is an untyped language, the property and method type information defined by the standard is still quite useful and is included in the reference pages in this section. This means that method and property synopses in this section use a syntax that is more like Java than like JavaScript. What follows is a sample reference page titled "Sample Entry" that demonstrates the structure of each reference page and explains how to interpret the information presented in each section. Even if you are already well familiar with the third edition of this book, take the time to read this page before diving into the DOM reference section.

Sample Entry

Availability

how to read DOM reference pages

Inherits from

Title and Short Description

Every reference entry begins with a four-part title block like that above. The entries are alphabetized by title. The short description, shown below the title, gives you a quick summary of the item documented in the entry; it can help you quickly decide if you're interested in reading the rest of the page.

Availability

The availability information is shown in the upper-right corner of the title block. This information tells you what level and what module of the DOM standard defines the interface or method. Since properties do not have their own reference pages, they do not have availability information. If the availability of a property is different from the availability of the interface that defines it, this fact is noted in the description of the property.

Inherits from

DOM interfaces can inherit properties and methods from other interfaces. If a DOM interface inherits from another interface, the inheritance hierarchy is shown in the lower-right corner of the title block. For example, the "Inherits from" information for the HTMLElement interface looks like this:

Node → Element → HTMLElement

This indicates that HTMLElement inherits from the Element interface, which in turn inherits from the Node interface. When you see this section, you may also want to look up the other listed interfaces.

Subinterfaces

This section contains the opposite of the "Inherits from" information: it lists any interfaces that inherit from this one. For example, the "Subinterfaces" section of the reference page

for the Element interface specifies that HTMLElement is a subinterface of Element and inherits Element's methods and properties.

Also Implements

The modular structure of the DOM standard means that some interfaces have been broken into multiple separate interfaces, so that implementations have to implement only the interfaces that are part of the modules they support. It is common for an object that implements one interface (such as Document) to also implement several other simple interfaces (such as DocumentCSS, DocumentEvent, and DocumentViews) that provide functionality specific to other modules. When an interface has minor interfaces that are intended to be implemented along with it, those minor interfaces are listed in this section.

Constants

Some DOM interfaces define a set of constants that serve as the values for a property or as the arguments to a method of that interface. The Node interface, for example, defines important constants to serve as the set of legal values for the `nodeType` property of all Document nodes. When an interface defines constants, they are listed and documented in this section. The listings include the type, the name, and the value (in that order) of each constant. See the "DOM Types" section for a discussion of the syntax used in these listings. Note that constants are static properties of the interface itself, not of instances of that interface.

Properties

If the reference page documents an interface, this section lists and documents the properties defined by that interface. Each entry in the list specifies the name and type of the property and may also include other keywords that provide additional information about the property. Note that in this Java-style syntax, the name of the property comes last, and all the information that precedes the name provides type and other information about the property. For example, the HTMLTableElement and HTMLTableCellElement interfaces define properties that include the following:

`HTMLTableCaptionElement caption`
: The `caption` property. It refers to an object of type HTMLTableCaptionElement.

`readonly HTMLCollection rows`
: The `rows` property. It refers to an HTMLCollection object and is read-only: you can query the value of the property, but you cannot set it.

`deprecated String align`
: The `align` property. It is a string, but it is deprecated and its use is discouraged.

`readonly long cellIndex`
: The `cellIndex` property. It is a long integer value (see the "DOM Types" section) and is read-only.

Methods

If the reference page documents an interface, this section lists the names of the interface's methods and provides a short description of each. Full documentation for each method is found in a separate reference page.

Synopsis

If the reference page documents a method, this section presents the method signature or synopsis. This section uses a Java-style syntax to specify (in order):

- The type of the method return value, or `void` if the method does not return anything.
- The name of the method.
- The type and name (in that order) of each argument of the method. These are presented as a comma-separated list of argument types and names within parentheses. If the method does not take any arguments, you simply see the parentheses: `( )`.
- The types of exceptions, if any, that the method can throw.

For example, the "Synopsis" section of the `Node.insertBefore( )` method looks like this:

```
Node insertBefore(Node newChild,
                  Node refChild)
    throws DOMException;
```

You can glean the following information from this synopsis: the name of the method is "insertBefore"; it returns a Node object; the first argument is a Node object and specifies the "newChild" (presumably the one to be inserted); the second argument is also a Node object and specifies the "refChild" (presumably the node before which the other is inserted); and the method may, in some circumstances, throw an exception of type DOMException.

The subsections that follow the synopsis provide additional information about the arguments, return value, and exceptions of the method. See also the "DOM Types" section for more information about the Java-style syntax used here to specify the types of method arguments.

Arguments

If a method has arguments, the "Synopsis" section is followed by an "Arguments" subsection that lists the names of the arguments and describes each one. Note that argument names are listed in *italics*, to indicate that they are not to be typed literally but instead represent some other value or JavaScript expression. To continue with the previous example, the "Arguments" section of `Node.insertBefore( )` looks like this:

newChild
: The node to be inserted into the tree. If it is a DocumentFragment, its children are inserted instead.

refChild
: The child of this node before which *newChild* is to be inserted. If this argument is `null`, *newChild* is inserted as the last child of this node.

Returns

The "Synopsis" section specifies the data type of the method's return value, and the "Returns" subsection provides additional information. If the method has no return value (i.e., if it is listed in the "Synopsis" section as returning `void`), this section is omitted.

Throws

This section explains the kinds of exceptions the method can throw and under what circumstances it throws them.

DOM Types

DOM reference pages use a Java-style syntax for specifying the types of constants, properties, method return values, and method arguments. This section provides more information about that syntax. Note that the reference pages themselves do not have "DOM Types" sections!

The general syntax is:

```
modifiers type name
```

The name of the constant, property, method, or method argument always comes last and is preceded by type and other information. The modifiers used in this reference section (note that these are not actually legal Java modifiers) are:

`readonly`
: Specifies that a property value can be queried but cannot be set.

`deprecated`
: Specifies that a property is deprecated and its use should be avoided.

`unsigned`
: Specifies that a numeric constant, property, return value, or method argument is unsigned; i.e., it may be zero or positive, but may not be negative.

The types of DOM constants, properties, method return values, and method arguments do not always correspond directly to the types supported by JavaScript. For example, some properties have a type of `short` which specifies a 16-bit integer. Although JavaScript only has a single numeric type, this reference section uses the DOM type simply because it provides more information about what range of numbers are legal. The DOM types you will encounter in this reference section are:

`String`
: A core JavaScript String object.

`Date`
: A core JavaScript Date object (this is not commonly used).

`boolean`
: A boolean value: `true` or `false`.

`short`
: A short (16-bit) integer. This type may have the `unsigned` modifier applied to it.

`long`
: A long (64-bit) integer. This type may have the `unsigned` modifier applied to it.

`float`
: A floating-point number. This type may not have the `unsigned` modifier applied to it.

`void`
: This type is used for method return values only; it indicates that the method does not return any value.

Any other type
: Any other types you see in this reference section are names of other DOM interfaces (for example, Document, DOMImplementation, Element, HTMLTableElement, and Node).

Description

Most reference pages contain a "Description" section, which is the basic description of the interface or method that is being documented. This is the heart of the reference page. If you are learning about an interface or method for the first time, you may want to skip directly to this section and then go back and look at previous sections such as "Synopsis," "Properties," and "Methods." If you are already familiar with an interface or method, you probably won't need to read this section and instead will just want to quickly look up some specific bit of information (such as the name of a property or the type of an argument from the "Properties" or "Arguments" sections).

In some pages, this section is no more than a short paragraph. In others, it may occupy a page or more. For some simple methods, the "Arguments," "Returns," and "Throws" sections document the method sufficiently by themselves, so the "Description" section is omitted.

Example

Reference pages for some commonly used interfaces and methods include an example in this section to illustrate typical usage of the interface or method. Most pages do not contain examples, however—you'll find those in first half of this book.

See Also

Most reference pages conclude with cross-references to related reference pages that may be of interest. Most of these cross-references are to other reference pages in this DOM reference section. Some are to individual property descriptions contained within an interface reference page, however, and others are to related reference pages in the client-side reference section or to chapters in the first two parts of the book.

Reference pages that document interfaces (but not those that document methods) may have additional paragraphs at the end of the "See Also" section. These are cross-references that show how the interface is used. A "Type of" paragraph lists properties whose values are objects that implement the interface. A "Passed to" paragraph lists methods that take an argument that implements the interface. A "Returned by" paragraph lists methods that return an object that implements the interface. These cross-references show how you can obtain an object of this interface and what you can do with it once you have obtained it.

AbstractView

DOM Level 2 Views

a window displaying a document

Also Implements

ViewCSS

If the DOM implementation supports the CSS module, any object that implements the AbstractView interface also implements the ViewCSS interface. For convenience, the method defined by the ViewCSS interface is listed under "Methods."

Properties

`readonly Document document`

The Document object that is displayed by this View object. This Document object also implements the DocumentView interface.

Methods

`getComputedStyle( )` *[DOM Level 2 CSS]*

This ViewCSS method returns a read-only CSSStyleDeclaration that represents the computed style information for a specific document element.

Description

In the DOM, a *view* is an object that displays a document in some way. The Window object of client-side JavaScript is such a view. This AbstractView interface is a very preliminary step toward standardizing some of the properties and methods of the Window object. It simply specifies that all View objects have a property named `document` that refers to the document they display. In addition, if an implementation supports CSS style sheets, all View objects also implement the ViewCSS interface and define a `getComputedStyle( )` method for determining how an element is actually rendered in the view.

The `document` property gives every view a reference to the document it displays. The reverse is true also: every document has a reference to the view that displays it. If a DOM implementation supports the View module, the object that implements the Document interface also implements the DocumentView interface. This DocumentView interface defines a `defaultView` property that refers to the window in which the document is displayed.

This interface has the word "Abstract" in its name to emphasize the fact that it is merely the beginning of a standardized window interface. In order to be useful, future levels of the DOM standard will have to introduce a new interface that extends AbstractView and adds other properties or methods.

See Also

Document.defaultView

Type of: Document.defaultView

AbstractView.getComputedStyle( )

DOM Level 2 CSS

retrieve the CSS styles used to render an element

Synopsis

```
CSSStyleDeclaration getComputedStyle(Element elt,
                                     String pseudoElt);
```

Arguments

`elt`

The document element whose style information is desired.

`pseudoElt`

The CSS pseudoelement, or `null` if there is none.

Returns

A read-only CSSStyleDeclaration object (which typically also implements the CSS2Properties interface) that specifies the style information used to render the specified element in this view. Any length values queried from this object are always absolute or pixel values, not relative or percentage values.

Description

An element in a document may obtain style information from an inline `style` attribute and from any number of style sheets in the style-sheet "cascade." Before the element can actually be displayed in a view, its style must be "computed" by extracting style information from the appropriate parts of the cascade.

This method allows access to those computed styles. By contrast, the `style` property of an element gives you access only to the inline styles of an element and tells you nothing about style-sheet attributes that apply to the element. Note that this method also provides a way to determine the actual pixel coordinates at which an element is rendered in this view.

`getComputedStyle( )` is actually defined by the ViewCSS interface. In any DOM implementation that supports the View and CSS modules, any object that implements AbstractView always implements ViewCSS also. So, for simplicity, this method has been listed with AbstractView.

In Internet Explorer, similar functionality is available through the nonstandard `currentStyle` property of each HTMLElement object.

See Also

CSS2Properties, CSSStyleDeclaration, HTMLElement.style

Attr

DOM Level 1 Core

an attribute of a document element

Node → Attr

Properties

`readonly String name`
: The name of the attribute.

`readonly Element ownerElement` *[DOM Level 2]*
: The Element object that contains this attribute, or `null` if the Attr object is not currently associated with any Element.

`readonly boolean specified`
: `true` if the attribute was explicitly specified in the document source or set by a script. `false` if the attribute was not explicitly specified but a default value is specified in the document's DTD.

`String value`
: The value of the attribute. When reading this property, the attribute value is returned as a string. When you set this property to a string, it automatically creates a Text node that contains the same text and makes that Text node the sole child of the Attr object.

Description

An Attr object represents an attribute of an Element node. Attr objects are associated with Element nodes but are not directly part of the document tree (and have a `null parentNode` property). You can obtain an Attr object through the `attributes` property of the Node interface or by calling the `getAttributeNode( )` method of the Element interface.

Attr objects are nodes, and the value of an Attr is represented by the child nodes of the Attr node. In HTML documents, this is simply a single Text node. In XML documents,

however, Attr nodes may have both Text and EntityReference children. The `value` property provides a shortcut for reading and writing the value of an attribute as a String.

In most cases, the easiest way to work with element attributes is with the `getAttribute()` and `setAttribute()` methods of the Element interface. These methods use strings for attribute names and values and avoid the use of Attr nodes altogether.

See Also

Element

Passed to: Element.removeAttributeNode(), Element.setAttributeNode(), Element.setAttributeNodeNS()

Returned by: Document.createAttribute(), Document.createAttributeNS(), Element.getAttributeNode(), Element.getAttributeNodeNS(), Element.removeAttributeNode(), Element.setAttributeNode(), Element.setAttributeNodeNS()

CDATASection

DOM Level 1 XML

a CDATA section in an XML document

Node → CharacterData → Text → CDATASection

Description

This infrequently used interface represents a CDATA section in an XML document. Programmers working with HTML documents never encounter nodes of this type and do not need to use this interface.

CDATASection is a subinterface of Text and does not define any properties or methods of its own. The textual content of the CDATA section is available through the `nodeValue` property inherited from Node or through the `data` property inherited from CharacterData. Although CDATASection nodes can often be treated in the same way as Text nodes, note that the `Node.normalize()` method does not merge adjacent CDATA sections.

See Also

CharacterData, Text

Returned by: Document.createCDATASection()

CharacterData

DOM Level 1 Core

common functionality for Text and Comment nodes

Node → CharacterData

Subinterfaces

Comment, Text

Properties

`String data`
: The text contained by this node.

`readonly unsigned long length`
: The number of characters contained by this node.

Methods

`appendData( )`
: Appends the specified string to the text contained by this node.

`deleteData( )`
: Deletes text from this node, starting with the character at the specified offset and continuing for the specified number of characters.

`insertData( )`
: Inserts the specified string into the text of this node at the specified character offset.

`replaceData( )`
: Replaces the characters starting at the specified character offset and continuing for the specified number of characters with the specified string.

`substringData( )`
: Returns a copy of the text starting at the specified character offset and continuing for the specified number of characters.

Description

CharacterData is the superinterface for Text and Comment nodes. Documents never contain CharacterData nodes; they contain only Text and Comment nodes. Since both of these node types have similar functionality, however, that functionality has been defined here so that both Text and Comment can inherit it.

See Also

Comment, Text

CharacterData.appendData( )

DOM Level 1 Core

append a string to a Text or Comment node

Synopsis

```
void appendData(String arg)
    throws DOMException;
```

Arguments

arg
: The string to be appended to the Text or Comment node.

Throws

This method throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if called on a node that is read-only.

Description

This method appends the string *arg* to the end of the `data` property for this node.

CharacterData.deleteData( )

DOM Level 1 Core

delete characters from a Text or Comment node

Synopsis

```
void deleteData(unsigned long offset,
                unsigned long count)
    throws DOMException;
```

Arguments

offset
: The position of the first character to be deleted.

count
: The number of characters to be deleted.

Throws

This method may throw a DOMException with one of the following code values:

INDEX_SIZE_ERR
: The *offset* or *count* argument is negative, or *offset* is greater than the length of the Text or Comment node.

NO_MODIFICATION_ALLOWED_ERR
: The node is read-only and may not be modified.

Description

This method deletes characters from this Text or Comment node, starting with the character at the position *offset* and continuing for *count* characters. If *offset* plus *count* is greater than the number of characters in the Text or Comment node, all characters from *offset* to the end of the string are deleted.

CharacterData.insertData( )

DOM Level 1 Core

insert a string into a Text or Comment node

Synopsis

```
void insertData(unsigned long offset,
                String arg)
    throws DOMException;
```

Arguments

offset
: The character position within the Text or Comment node at which the string is to be inserted.

arg
: The string to insert.

Throws

This method may throw a DOMException with one of the following code values in the following circumstances:

`INDEX_SIZE_ERR`
: *offset* is negative or greater than the length of the Text or Comment node.

`NO_MODIFICATION_ALLOWED_ERR`
: The node is read-only and may not be modified.

Description

This method inserts the specified string *arg* into the text of a Text or Comment node at the specified position *offset*.

CharacterData.replaceData( )

DOM Level 1 Core

replace characters of a Text or Comment node with a string

Synopsis

```
void replaceData(unsigned long offset,
                 unsigned long count,
                 String arg)
    throws DOMException;
```

Arguments

offset
: The character position within the Text or Comment node at which the replacement is to begin.

count
: The number of characters to be replaced.

arg
: The string that replaces the characters specified by *offset* and *count*.

Throws

This method may throw a DOMException with one of the following code values in the following circumstances:

`INDEX_SIZE_ERR`
: *offset* is negative or greater than the length of the Text or Comment node, or *count* is negative.

`NO_MODIFICATION_ALLOWED_ERR`
: The node is read-only and may not be modified.

Description

This method replaces *count* characters starting at position *offset* with the contents of the string *arg*. If the sum of *offset* and *count* is greater than the length of the Text or Comment node, all characters from *offset* on are replaced.

CharacterData.substringData( )

DOM Level 1 Core

extract a substring from a Text or Comment node

Synopsis

```
String substringData(unsigned long offset,
                     unsigned long count)
    throws DOMException;
```

Arguments

offset
: The position of the first character to be returned.

count
: The number of characters in the substring to be returned.

Returns

A string that consists of *count* characters of the Text or Comment node starting with the character at position *offset*.

Throws

This method may throw a DOMException with one of the following code values:

INDEX_SIZE_ERR
: *offset* is negative or greater than the length of the Text or Comment node, or *count* is negative.

DOMSTRING_SIZE_ERR
: The specified range of text is too long to fit into a string in the browser's JavaScript implementation.

Description

This method extracts the substring that starts at position *offset* and continues for *count* characters from the text of a Text or Comment node. This method is useful only when the amount of text contained by the node is larger than the maximum number of characters that can fit in a string in a browser's JavaScript implementation. In this case, a JavaScript program cannot use the data property of the Text or Comment node directly and must instead work with shorter substrings of the node's text. This situation is unlikely to arise in practice.

Comment

DOM Level 1 Core

an HTML or XML comment

Node → CharacterData → Comment

Description

A Comment node represents a comment in an HTML or XML document. The content of the comment (i.e., the text between <!-- and -->) is available through the data property inherited from the CharacterData interface or through the nodeValue property inherited from the Node interface. This content may be manipulated using the various methods inherited from CharacterData.

See Also

CharacterData

Returned by: Document.createComment( )

Counter

DOM Level 2 CSS

a CSS counter( ) or counters( ) specification

Properties

`readonly String identifier`
: The name of the counter.

`readonly String listStyle`
: The list style for the counter.

`readonly String separator`
: The separator string for nested counters.

Description

This interface represents a CSS `counter( )` or `counters( )` value. Consult a CSS reference for more information.

See Also

Returned by: CSSPrimitiveValue.getCounterValue( )

CSS2Properties

DOM Level 2 CSS2

convenience properties for all CSS2 attributes

Properties

This interface defines a large number of properties: one property for each CSS attribute defined by the CSS2 specification. The property names correspond closely to the CSS attribute names, with minor changes required to avoid syntax errors in JavaScript. Multiword attributes that contain hyphens, such as "font-family," are written without hyphens in JavaScript, and each word after the first is capitalized: `fontFamily`. Also, the "float" attribute conflicts with the reserved word `float`, so it translates to the property `cssFloat`.

The complete set of properties is listed in the following table. Since the properties correspond directly to CSS attributes, no individual documentation is given for each property. See a CSS reference, such as *Cascading Style Sheets: The Definitive Guide*, by Eric A. Meyer (O'Reilly), for the meaning and legal values of each. All of the properties are strings. Setting any of these properties may throw the same exceptions, for the same reasons as a call to `CSSStyleDeclaration.setProperty( )`.

`azimuth`	`background`	`backgroundAttachment`	`backgroundColor`
`backgroundImage`	`backgroundPosition`	`backgroundRepeat`	`border`

borderBottom	borderBottomColor	borderBottomStyle	borderBottomWidth
borderCollapse	borderColor	borderLeft	borderLeftColor
borderLeftStyle	borderLeftWidth	borderRight	borderRightColor
borderRightStyle	borderRightWidth	borderSpacing	borderStyle
borderTop	borderTopColor	borderTopStyle	borderTopWidth
borderWidth	bottom	captionSide	clear
clip	color	content	counterIncrement
counterReset	cssFloat	cue	cueAfter
cueBefore	cursor	direction	display
elevation	emptyCells	font	fontFamily
fontSize	fontSizeAdjust	fontStretch	fontStyle
fontVariant	fontWeight	height	left
letterSpacing	lineHeight	listStyle	listStyleImage
listStylePosition	listStyleType	margin	marginBottom
marginLeft	marginRight	marginTop	markerOffset
marks	maxHeight	maxWidth	minHeight
minWidth	orphans	outline	outlineColor
outlineStyle	outlineWidth	overflow	padding
paddingBottom	paddingLeft	paddingRight	paddingTop
page	pageBreakAfter	pageBreakBefore	pageBreakInside
pause	pauseAfter	pauseBefore	pitch
pitchRange	playDuring	position	quotes
richness	right	size	speak
speakHeader	speakNumeral	speakPunctuation	speechRate
stress	tableLayout	textAlign	textDecoration
textIndent	textShadow	textTransform	top
unicodeBidi	verticalAlign	visibility	voiceFamily
volume	whiteSpace	widows	width
wordSpacing	zIndex		

Description

This interface defines one property for each CSS attribute defined by the CSS2 specification. If the DOM implementation supports this interface (which is part of the "CSS2" feature), all CSSStyleDeclaration objects also implement CSS2Properties. Reading one of the properties defined by this interface is equivalent to calling `getPropertyValue( )` for the corresponding CSS attribute, and setting the value of one of these properties is equivalent to calling `setProperty( )` for the corresponding attribute. The properties defined by CSS2Properties include properties that correspond to CSS shortcut attributes, and CSS2Properties handles these shortcut properties correctly.

See Also

CSSStyleDeclaration

CSSCharsetRule

DOM Level 2 CSS

an @charset rule in a CSS style sheet

CSSRule → CSSCharsetRule

Properties

`String encoding`
: The character encoding specified by the `@charset` rule. If you set this property to an illegal value, a DOMException with a `code` of `SYNTAX_ERR` is thrown. If the rule or style sheet is read-only, an attempt to set this property throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR`.

Description

This interface represents an `@charset` rule in a CSS style sheet. Consult a CSS reference for details.

CSSFontFaceRule

DOM Level 2 CSS

an @font-face rule in a CSS style sheet

CSSRule → CSSFontFaceRule

Properties

`readonly CSSStyleDeclaration style`
: The set of styles for this rule.

Description

This interface represents an `@font-face` rule in a CSS style sheet. Consult a CSS reference for details.

CSSImportRule

DOM Level 2 CSS

an @import rule in a CSS style sheet

CSSRule → CSSImportRule

Properties

`readonly String href`
: The URL of the imported style sheet. The value of this property does not include the "url()" delimiter around the URL value.

`readonly MediaList media`
: A list of media types to which the imported style sheet applies.

`readonly CSSStyleSheet styleSheet`
: The CSSStyleSheet object that represents the imported style sheet, or `null` if the style sheet has not yet been loaded or if the style sheet was not loaded because, for example, the media type did not apply.

Description

This interface represents an `@import` rule in a CSS style sheet. The `styleSheet` property represents the imported style sheet.

CSSMediaRule

DOM Level 2 CSS

an @media rule in a CSS style sheet

CSSRule → CSSMediaRule

Properties

`readonly CSSRuleList cssRules`
: An array (technically, a CSSRuleList) of all the rules nested within this `@media` rule block.

`readonly MediaList media`
: The list of media types to which the nested rules apply.

Methods

`deleteRule( )`
: Deletes the nested rule at the specified position.

`insertRule( )`
: Inserts a new rule at the specified position within this `@media` rule block.

Description

This interface represents an `@media` rule, and all of its nested rules, in a CSS style sheet. It defines methods that allow you to insert and delete nested rules. Consult a CSS reference for details.

CSSMediaRule.deleteRule( )

DOM Level 2 CSS

delete a rule in an @media block

Synopsis

```
void deleteRule(unsigned long index)
    throws DOMException;
```

Arguments

index
: The position within the `@media` rule block of the rule to be deleted.

Throws

This method throws a DOMException with one of the following `code` values in the following circumstances:

`INDEX_SIZE_ERR`
: *index* is negative or is greater than or equal to the number of rules in `cssRules`.

`NO_MODIFICATION_ALLOWED_ERR`
: The rule is read-only.

Description

This method deletes the rule at the specified position in the `cssRules` array.

CSSMediaRule.insertRule( )

DOM Level 2 CSS

insert a new rule into an @media block

Synopsis

```
unsigned long insertRule(String rule,
                         unsigned long index)
    throws DOMException;
```

Arguments

rule

The complete, parseable CSS string representation of the rule to be added.

index

The position at which the new rule is to be inserted into the `cssRules` array, or the `cssRules.length` to append the new rule at the end of the array.

Returns

The value of the *index* argument.

Throws

This method throws a DOMException with one of the following `code` values in the following circumstances:

`HIERARCHY_REQUEST_ERR`

CSS syntax does not allow the specified *rule* at the specified position.

`INDEX_SIZE_ERR`

index is negative or greater than `cssRules.length`.

`NO_MODIFICATION_ALLOWED_ERR`

This `@media` rule and its `cssRules` array are read-only.

`SYNTAX_ERR`

The specified *rule* contains a syntax error.

Description

This method inserts the specified *rule* into the `cssRules` array at the specified *index*.

CSSPageRule

DOM Level 2 CSS

an @page rule in a CSS style sheet

CSSRule → CSSPageRule

Properties

`String selectorText`

The page selector text for this rule. Setting this property to an illegal value throws a DOMException with a `code` of `SYNTAX_ERR`. Setting this property when the rule is read-only throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR`.

`readonly CSSStyleDeclaration style`

The set of styles for this rule.

Description

This interface represents an @page rule in a CSS style sheet, which is typically used to specify the page layout for printing. Consult a CSS reference for details.

CSSPrimitiveValue

DOM Level 2 CSS

a single CSS style value

CSSValue → CSSPrimitiveValue

Constants

The following constants are the legal values for the primitiveType property. They specify the type of the value and, for numeric values, the units in which the value is represented.

`unsigned short CSS_UNKNOWN = 0`
: The value is not recognized, and the implementation does not know how to parse it. The textual representation of the value is available through the cssText property.

`unsigned short CSS_NUMBER = 1`
: A unitless number. Query with getFloatValue( ).

`unsigned short CSS_PERCENTAGE = 2`
: A percentage. Query with getFloatValue( ).

`unsigned short CSS_EMS = 3`
: A relative length measured in ems (the height of the current font). Query with getFloatValue( ).

`unsigned short CSS_EXS = 4`
: A relative length measured in exs (the "x-height" of the current font). Query with getFloatValue( ).

`unsigned short CSS_PX = 5`
: A length measured in pixels. Query with getFloatValue( ). Pixel lengths are relative measurements, in the sense that their size depends on the display resolution, and they cannot be converted to inches, millimeters, points, or other absolute lengths. However, pixels are also one of the most commonly used units, and they are treated as absolute values for the purposes of AbstractView.getComputedStyle( ), for example.

`unsigned short CSS_CM = 6`
: An absolute length measured in centimeters. Query with getFloatValue( ).

`unsigned short CSS_MM = 7`
: An absolute length measured in millimeters. Query with getFloatValue( ).

`unsigned short CSS_IN = 8`
: An absolute length measured in inches. Query with getFloatValue( ).

`unsigned short CSS_PT = 9`
: An absolute length measured in points (1/72 of an inch). Query with getFloatValue( ).

`unsigned short CSS_PC = 10`
: An absolute length measured in picas (12 points). Query with getFloatValue( ).

`unsigned short CSS_DEG = 11`
: An angle measured in degrees. Query with getFloatValue( ).

`unsigned short CSS_RAD = 12`
: An angle measured in radians. Query with getFloatValue( ).

`unsigned short CSS_GRAD = 13`
An angle measured in grads. Query with `getFloatValue( )`.

`unsigned short CSS_MS = 14`
A time measured in milliseconds. Query with `getFloatValue( )`.

`unsigned short CSS_S = 15`
A time measured in seconds. Query with `getFloatValue( )`.

`unsigned short CSS_HZ = 16`
A frequency measured in hertz. Query with `getFloatValue( )`.

`unsigned short CSS_KHZ = 17`
A frequency measured in kilohertz. Query with `getFloatValue( )`.

`unsigned short CSS_DIMENSION = 18`
A unitless dimension. Query with `getFloatValue( )`.

`unsigned short CSS_STRING = 19`
A string. Query with `getStringValue( )`.

`unsigned short CSS_URI = 20`
A URI. Query with `getStringValue( )`.

`unsigned short CSS_IDENT = 21`
An identifier. Query with `getStringValue( )`.

`unsigned short CSS_ATTR = 22`
An attribute function. Query with `getStringValue( )`.

`unsigned short CSS_COUNTER = 23`
A counter. Query with `getCounterValue( )`.

`unsigned short CSS_RECT = 24`
A rectangle. Query with `getRectValue( )`.

`unsigned short CSS_RGBCOLOR = 25`
A color. Query with `getRGBColorValue( )`.

Properties

`readonly unsigned short primitiveType`
The type of this value. This property holds one of the constants defined in the previous section.

Methods

`getCounterValue( )`
For values of type `CSS_COUNTER`, returns the Counter object that represents the value.

`getFloatValue( )`
Returns a numeric value, converting it, if necessary, to the specified units.

`getRectValue( )`
For values of type `CSS_RECT`, returns the Rect object that represents the value.

`getRGBColorValue( )`
For values of type `CSS_RGBCOLOR`, returns the RGBColor object that represents the value.

`getStringValue( )`
: Returns the value as a string.

`setFloatValue( )`
: Sets a numeric value to the specified number of the specified units.

`setStringValue( )`
: Sets a string value to the specified string of the specified type.

Description

This subinterface of CSSValue represents a single CSS value. Contrast it with the CSSValueList interface, which represents a list of CSS values. The word "primitive" in the name of this interface is misleading; this interface can represent some complex types of CSS values, such as counters, rectangles, and colors.

The `primitiveType` property holds one of the previously defined constants and specifies the type of the value. The various methods defined by this interface allow you to query values of various types and also to set numeric and string values.

See Also

Counter, CSSValue, CSSValueList, Rect, RGBColor

Type of: RGBColor.blue, RGBColor.green, RGBColor.red, Rect.bottom, Rect.left, Rect.right, Rect.top

CSSPrimitiveValue.getCounterValue( )

DOM Level 2 CSS

return a Counter value

Synopsis

```
Counter getCounterValue( )
    throws DOMException;
```

Returns

The Counter object that represents the value of this CSSPrimitiveValue.

Throws

This method throws a DOMException with a `code` of `INVALID_ACCESS_ERR` if the `primitiveType` property is not `CSS_COUNTER`.

Description

This method returns a Counter object that represents a CSS counter. There is no corresponding `setCounterValue( )`, but you can modify the value by setting the properties of the returned Counter object.

CSSPrimitiveValue.getFloatValue( ) DOM Level 2 CSS

get a numeric value, possibly converting units

Synopsis

```
float getFloatValue(unsigned short unitType)
    throws DOMException;
```

Arguments

unitType
: One of the CSSPrimitiveValue type constants that specifies the desired units for the returned value.

Returns

The floating-point numeric value of this CSSPrimitiveValue, expressed in the specified units.

Throws

This method throws a DOMException with a `code` of `INVALID_ACCESS_ERR` if this CSSPrimitiveValue holds a non-numeric value, or if the value cannot be converted to the requested type of units. (See the next section for more about unit conversion.)

Description

For CSSPrimitiveValue objects that hold numeric values, this method converts those values to the specified units and returns the converted values.

Only certain types of unit conversions are allowed. Lengths may be converted to lengths, angles to angles, times to times, and frequencies to frequencies. Obviously, however, a length measured in millimeters cannot be converted to a frequency measured in kilohertz. Also, not all lengths can be converted. Relative lengths (lengths measured in ems, exs, or pixels) can be converted to other relative lengths but cannot be converted to absolute lengths. Similarly, absolute lengths cannot be converted to relative lengths. Finally, percentage values cannot be converted to any other unit type, except for color percentage values, which express a percentage of 255 and can be converted to the `CSS_NUMBER` type.

CSSPrimitiveValue.getRectValue( ) DOM Level 2 CSS

return a Rect value

Synopsis

```
Rect getRectValue( )
    throws DOMException;
```

Returns

The Rect object that represents the value of this CSSPrimitiveValue.

Throws

This method throws a DOMException with a code of INVALID_ACCESS_ERR if the primitiveType property is not CSS_RECT.

Description

This method returns a Rect object that represents a CSS rectangle. There is no corresponding setRectValue( ) method, but you can modify the value by setting the properties of the returned Rect object.

CSSPrimitiveValue.getRGBColorValue( )

DOM Level 2 CSS

get the RGBColor value

Synopsis

```
RGBColor getRGBColorValue( )
    throws DOMException;
```

Returns

The RGBColor object that represents the value of this CSSPrimitiveValue.

Throws

This method throws a DOMException with a code of INVALID_ACCESS_ERR if the primitiveType property is not CSS_RGBCOLOR.

Description

This method returns an RGBColor object that represents a color. There is no corresponding setRGBColorValue( ) method, but you can modify the value by setting the properties of the returned RGBColor object.

CSSPrimitiveValue.getStringValue( )

DOM Level 2 CSS

query a CSS string value

Synopsis

```
String getStringValue( )
    throws DOMException;
```

Returns

The string value of this CSSPrimitiveValue.

Throws

This method throws a DOMException with a code of INVALID_ACCESS_ERR if the primitiveType property is not CSS_STRING, CSS_URI, CSS_IDENT, or CSS_ATTR.

CSSPrimitiveValue.setFloatValue( ) DOM Level 2 CSS

set the numeric value

Synopsis

```
void setFloatValue(unsigned short unitType,
                   float floatValue)
    throws DOMException;
```

Arguments

unitType
: One of the CSSPrimitiveValue constants that specifies the numeric type units for this value.

floatValue
: The new value (measured in *unitType* units).

Throws

This method throws a DOMException with a code of NO_MODIFICATION_ALLOWED_ERR if the CSS attribute with which this value is associated is read-only. It throws a DOMException with a code of INVALID_ACCESS_ERR if that CSS attribute does not allow numeric values or does not allow values with the specified *unitType*.

Description

This method specifies the unit type and numeric value for this CSSPrimitiveValue.

CSSPrimitiveValue.setStringValue( ) DOM Level 2 CSS

set the string value

Synopsis

```
void setStringValue(unsigned short stringType,
                    String stringValue)
    throws DOMException;
```

Arguments

stringType
: The type of the string being set. This must be one of the CSSPrimitiveValue constants CSS_STRING, CSS_URI, CSS_IDENT, or CSS_ATTR.

stringValue
: The new string value to be set.

Throws

This method throws a DOMException with a code of NO_MODIFICATION_ALLOWED_ERR if the CSS attribute with which this value is associated is read-only. It throws a DOMException with a code of INVALID_ACCESS_ERR if that CSS attribute does not allow string values or does not allow values with the specified *stringType*.

Description

This method sets the string value and string type for this CSSPrimitiveValue.

CSSRule

DOM Level 2 CSS

a rule in a CSS style sheet

Subinterfaces

CSSCharsetRule, CSSFontFaceRule, CSSImportRule, CSSMediaRule, CSSPageRule, CSSStyleRule, CSSUnknownRule

Constants

These constants represent the various types of rules that may appear in a CSS style sheet. They are the legal values of the type property, and they specify which of the above subinterfaces this object implements.

```
unsigned short UNKNOWN_RULE = 0;    // CSSUnknownRule
unsigned short STYLE_RULE = 1;      // CSSStyleRule
unsigned short CHARSET_RULE = 2;    // CSSCharsetRule
unsigned short IMPORT_RULE = 3;     // CSSImportRule
unsigned short MEDIA_RULE = 4;      // CSSMediaRule
unsigned short FONT_FACE_RULE = 5;  // CSSFontFaceRule
unsigned short PAGE_RULE = 6;       // CSSPageRule
```

Properties

`String cssText`

The textual representation of the rule. If you set this property, it may throw a DOMException with one of the following code values for one of the following reasons:

`HIERARCHY_REQUEST_ERR`

The specified rule is not legal at this location in the style sheet.

`INVALID_MODIFICATION_ERR`

The new value of the property is a rule of a different type than the original value.

`NO_MODIFICATION_ALLOWED_ERR`

The rule or the style sheet that contains it is read-only.

`SYNTAX_ERR`

The specified string is not legal CSS syntax.

`readonly CSSRule parentRule`

The containing rule of this rule, or `null` if this rule does not have a parent. An example of a CSS rule with a parent is a style rule within an `@media` rule.

`readonly CSSStyleSheet parentStyleSheet`

The CSSStyleSheet object that contains this rule.

`readonly unsigned short type`

The type of CSS rule this object represents. The legal values for this property are the previously listed constants. This CSSRule interface is never implemented directly, and the value of this property specifies which more specific subinterface is implemented by this object.

Description

This interface defines properties that are common to all types of rules in CSS style sheets. No object directly implements this interface; instead, they implement one of the more specific subinterfaces listed earlier. The most important subinterface is probably CSSStyleRule, which describes a CSS rule that defines a document style.

See Also

CSSStyleRule

Type of: CSSRule.parentRule, CSSStyleDeclaration.parentRule, CSSStyleSheet.ownerRule

Returned by: CSSRuleList.item( )

CSSRuleList

DOM Level 2 CSS

an array of CSSRule objects

Properties

`readonly unsigned long length`
: The number of CSSRule objects in this CSSRuleList array.

Methods

`item( )`
: Returns the CSSRule object at the specified position. Instead of explicitly calling this method, JavaScript allows you to simply treat the CSSRuleList object as an array and to index it using standard square-bracket array notation. If the specified index is too large, this method returns `null`.

Description

This interface defines a read-only ordered list (i.e., an array) of CSSRule objects. The `length` property specifies the number of rules in the list, and the `item( )` method allows you to retrieve the rule at a specified position. In JavaScript, CSSRuleList objects behave like JavaScript arrays, and you can query an element from the list using square-bracket array notation instead of calling the `item( )` method. (Note, however, that you cannot assign new nodes into a CSSRuleList using square brackets.)

See Also

Type of: CSSMediaRule.cssRules, CSSStyleSheet.cssRules

CSSRuleList.item( )

DOM Level 2 CSS

get the CSSRule at the specified position

Synopsis

```
CSSRule item(unsigned long index);
```

Arguments

index
: The position of the rule to retrieve.

Returns

The CSSRule object at the specified position, or `null` if *index* is not a valid position.

CSSStyleDeclaration

DOM Level 2 CSS

a set of CSS style attributes and their values

Also Implements

If the implementation supports the "CSS2" feature in addition to the "CSS" feature (as most web browsers do), all objects that implement this interface also implement the CSS2Properties interface. CSS2Properties provides commonly used shortcut properties for setting and querying the values of CSS attributes. See "CSS2Properties" for details.

Properties

`String cssText`
: The textual representation of the style attributes and their values. This property consists of the complete text of the style rule, minus the element selector and the curly braces that surround the attributes and values. Setting this property to an illegal value throws a DOMException with a `code` of `SYNTAX_ERR`. Setting it for a read-only style sheet or rule throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR`.

`readonly unsigned long length`
: The number of style attributes in this style declaration.

`readonly CSSRule parentRule`
: The CSSRule object that contains this CSSStyleDeclaration, or `null` if this style declaration is not part of a CSS rule (such as for CSSStyleDeclaration objects that represent inline HTML `style` attributes).

Methods

`getPropertyCSSValue( )`
: Returns a CSSValue object that represents the value of the named CSS attribute, or `null` if that attribute is not explicitly set in this style declaration block or if the named style is a "shortcut" attribute.

`getPropertyPriority( )`
: Returns the string "important" if the named CSS attribute is explicitly set in this declaration block and has the `!important` priority qualifier specified. If the attribute is not specified, or has no priority, returns the empty string.

`getPropertyValue( )`
: Returns the value of the named CSS attribute as a string. Returns the empty string if the attribute is not specified in this declaration block.

`item( )`
: Returns the name of the CSS attribute at the specified position in this style declaration block. In JavaScript, the CSSStyleDeclaration object can be treated as an array and indexed using square brackets instead. See also the `length` property.

`removeProperty( )`
: Deletes a named CSS attribute from this declaration block.

`setProperty( )`
: Sets a named CSS attribute to the specified string value and priority for this declaration block.

Description

This attribute represents a CSS *style declaration block*: a set of CSS attributes and their values, separated from each other by semicolons. The style declaration block is the portion of a style rule within curly braces in a CSS style sheet. The value of the HTML `style` attribute also constitutes a style declaration block.

The `item( )` method and the `length` property allow you to loop through the names of all CSS attributes specified in this declaration block. In JavaScript, you can also simply treat the CSSStyleDeclaration object as an array and index it using square-bracket notation instead of calling the `item( )` method explicitly. Once you have the names of the CSS attributes specified in this declaration, you can use other methods of this interface to query the values of those attributes. `getPropertyValue( )` returns the value as a string, and `getPropertyCSSValue( )` returns the attribute value as a CSSValue object. (Note that the DOM API refers to CSS style attributes as "properties." I use the term "attributes" here to avoid confusing them with JavaScript object properties.)

In most web browsers, every object that implements CSSStyleDeclaration also implements the CSS2Properties interface, which defines an object property for each CSS attribute defined by the CSS2 specification. You can read and write the values of these convenience properties instead of calling `getPropertyValue( )` and `setProperty( )`.

See Also

CSS2Properties

Type of: CSSFontFaceRule.style, CSSPageRule.style, CSSStyleRule.style, HTMLElement.style

Returned by: Document.getOverrideStyle( ), AbstractView.getComputedStyle( )

CSSStyleDeclaration.getPropertyCSSValue( ) DOM Level 2 CSS

return a CSS attribute value as an object

Synopsis

```
CSSValue getPropertyCSSValue(String propertyName);
```

Arguments

propertyName
The name of the desired CSS attribute.

Returns

A CSSValue object that represents the value of the named attribute if it is explicitly specified in this style declaration, or `null` if the named attribute is not specified. This method also returns `null` if *propertyName* specifies a CSS shorthand attribute, since shorthand attributes specify more than one value and cannot be represented with CSSValue objects.

CSSStyleDeclaration.getPropertyPriority( ) DOM Level 2 CSS

get the priority of a CSS attribute

Synopsis

```
String getPropertyPriority(String propertyName);
```

Arguments

propertyName
The name of the CSS attribute.

Returns

The string "important" if the named CSS attribute is explicitly specified in this declaration block and has the `!important` priority modifier. Returns the empty string otherwise.

CSSStyleDeclaration.getPropertyValue( ) DOM Level 2 CSS

get the value of a CSS attribute as a string

Synopsis

```
String getPropertyValue(String propertyName);
```

Arguments

propertyName
The name of the CSS attribute whose value is desired.

Returns

The string value of the named CSS attribute, or the empty string if that attribute is not explicitly set in this declaration block.

Description

This method returns the value of the named CSS attribute as a string. Unlike `getPropertyCSSValue( )`, this method works with shortcut attributes as well as regular attributes. See also the various convenience properties of the CSS2Properties interface.

CSSStyleDeclaration.item( ) DOM Level 2 CSS

get the CSS attribute name at the specified position

Synopsis

```
String item(unsigned long index);
```

Arguments

index
The position of the desired CSS attribute name.

Returns

The name of the CSS attribute at *index*, or the empty string if *index* is negative or greater than or equal to the `length` property.

Description

The CSSStyleDeclaration interface represents a collection of CSS style attributes and their values. This method allows you to query the name of the CSS attribute by position and, in conjunction with the `length` property, allows you to iterate through the set of CSS attributes specified in this style declaration. Note that the order of CSS attributes as returned by this method does not necessarily correspond to the order in which they appear in the document or style sheet source.

As an alternative to this `item( )` method, JavaScript allows you to simply treat a CSSStyleDeclaration object as an array of CSS attribute names and use standard square-bracket array syntax to obtain the attribute name at a specified position.

CSSStyleDeclaration.removeProperty( ) DOM Level 2 CSS

delete a CSS attribute specification

Synopsis

```
String removeProperty(String propertyName)
    throws DOMException;
```

Arguments

propertyName
The name of the CSS attribute to be deleted.

Returns

The value of the named CSS attribute as a string, or the empty string if the named attribute is not explicitly specified in this style declaration.

Throws

If this style declaration is read-only, this method throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR`.

Description

This method deletes a named attribute from this style declaration block and returns the value of the attribute.

CSSStyleDeclaration.setProperty( ) DOM Level 2 CSS

set a CSS style attribute

Synopsis

```
void setProperty(String propertyName,
                 String value,
                 String priority)
    throws DOMException;
```

Arguments

propertyName
: The name of the CSS attribute to set.

value
: The new value of the attribute, as a string.

priority
: The new priority of the attribute. This argument should be "important" if the attribute specification is `!important`; otherwise, it should be the empty string.

Throws

This method throws a DOMException with a `code` of `SYNTAX_ERR` if the specified *value* argument is malformed. It throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if the style declaration or the attribute being set is read-only.

Description

This method adds the named CSS attribute with its value and priority to this style declaration, or, if the declaration already contains a value for the named attribute, it simply sets the value and priority for that existing attribute.

Using `setProperty( )` to add a new CSS attribute to a style declaration may insert the new attribute at any position and may, in fact, totally shuffle the order of all existing attributes. Therefore, you should not use `setProperty( )` while you are iterating through the set of attribute names with the `item( )` method.

CSSStyleRule DOM Level 2 CSS

a style rule in a CSS style sheet — CSSRule → CSSStyleRule

Properties

`String selectorText`
: The selector text that specifies the document elements this style rule applies to. Setting this property raises a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if

the rule is read-only, or a code of SYNTAX_ERR if the new value does not follow CSS syntax rules.

`readonly CSSStyleDeclaration style`
The style values that should be applied to elements specified by `selectorText`.

Description

This interface represents a style rule in a CSS style sheet. Style rules are the most common and important kinds of rules in style sheets: they specify style information that is to be applied to a specific set of document elements. `selectorText` is the string representation of the element selector for this rule, and `style` is a CSSStyleDeclaration object that represents the set of style names and values to apply to the selected elements.

See Also

CSSStyleDeclaration

CSSStyleSheet

DOM Level 2 CSS

a CSS style sheet

StyleSheet → CSSStyleSheet

Properties

`readonly CSSRuleList cssRules`
An array (technically, a CSSRuleList) of the CSSRule objects that comprise the style sheet. This includes all at-rules in addition to the actual style rules.

`readonly CSSRule ownerRule`
If this style sheet was imported by an `@import` rule in another style sheet, this property holds the CSSImportRule object that represents that `@import` rule. Otherwise, it is `null`. When this property is non-`null`, the inherited `ownerNode` property is `null`.

Methods

`deleteRule( )`
Deletes the rule at the specified position.

`insertRule( )`
Inserts a new rule at the specified position.

Description

This interface represents a CSS style sheet. The `cssRules` property lists the rules contained in the style sheet, and the `insertRule( )` and `deleteRule( )` methods allow you to add and delete rules from that list.

See Also

StyleSheet

Type of: CSSImportRule.styleSheet, CSSRule.parentStyleSheet

Returned by: DOMImplementation.createCSSStyleSheet( )

CSSStyleSheet.deleteRule( ) DOM Level 2 CSS

delete a rule from a style sheet

Synopsis

```
void deleteRule(unsigned long index)
    throws DOMException;
```

Arguments

index
: The index within the cssRules array of the rule to be deleted.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is negative or greater than or equal to cssRules.length. It throws a DOMException with a code of NO_MODIFICATION_ALLOWED_ERR if this style sheet is read-only.

Description

This method deletes the rule at the specified *index* from the cssRules array.

CSSStyleSheet.insertRule( ) DOM Level 2 CSS

insert a rule into a style sheet

Synopsis

```
unsigned long insertRule(String rule,
                         unsigned long index)
    throws DOMException;
```

Arguments

rule
: The complete, parseable text representation of the rule to be added to the style sheet. For style rules, this includes both the element selector and the style information.

index
: The position in the cssRules array at which the rule is to be inserted or appended.

Returns

The value of the *index* argument.

Throws

This method throws a DOMException with one of the following code values in the following circumstances:

HIERARCHY_REQUEST_ERR
: CSS syntax does not allow the specified rule at the specified location.

INDEX_SIZE_ERR
: *index* is negative or greater then cssRules.length.

`NO_MODIFICATION_ALLOWED_ERR`
: The style sheet is read-only.

`SYNTAX_ERR`
: The specified *`rule`* text contains a syntax error.

Description

This method inserts (or appends) a new CSS *`rule`* at the specified *`index`* of the `cssRules` array of this style sheet.

CSSUnknownRule

DOM Level 2 CSS

an unrecognized rule in a CSS style sheet

CSSRule → CSSUnknownRule

Description

This interface represents a rule in a CSS style sheet that the browser did not recognize and could not parse (typically because it is defined by a version of the CSS standard that the browser does not support). Note that this interface does not define any properties or methods of its own. The text of the unrecognized rule is available through the inherited `cssText` property.

CSSValue

DOM Level 2 CSS

the value of a CSS style attribute

Subinterfaces

CSSPrimitiveValue, CSSValueList

Constants

The following constants specify the valid values for the `cssValueType` property:

`unsigned short CSS_INHERIT = 0`
: This constant represents the special value "inherit", which means that the actual value of the CSS style attribute is inherited. The `cssText` property is "inherit" in this case.

`unsigned short CSS_PRIMITIVE_VALUE = 1`
: The value is a primitive value. This CSSValue object also implements the more specific CSSPrimitiveValue subinterface.

`unsigned short CSS_VALUE_LIST = 2`
: The value is a compound value consisting of a list of values. This CSSValue object also implements the more specific CSSValueList subinterface and behaves as an array of CSSValue objects.

`unsigned short CSS_CUSTOM = 3`
: This constant is defined to allow extensions to the CSS object model. It specifies that this CSSValue represents a value of some type that is not defined by the CSS or DOM standards. If you are working with an implementation that supports such extensions, the CSSValue object may also implement some other interface (such as the SVGColor interface defined by the Scalable Vector Graphics standard) that you can use.

Properties

`String cssText`
: The textual representation of the value. Setting this property may throw a DOMException. A `code` of `SYNTAX_ERR` indicates that the new value does not follow legal CSS syntax. A `code` of `INVALID_MODIFICATION_ERR` specifies that you tried to set a value of a different type than the original value. A `code` of `NO_MODIFICATION_ALLOWED_ERR` indicates that the value is read-only.

`readonly unsigned short cssValueType`
: The kind of value this object represents. The four legal values of this property are defined by the previously listed constants.

Description

This interface represents the value of a CSS attribute. The `cssText` property gives the value in textual form. If the `cssValueType` property is `CSSValue.CSS_PRIMITIVE_VALUE`, this CSSValue object also implements the more specific CSSPrimitiveValue interface. If `cssValueType` is `CSSValue.CSS_VALUE_LIST`, this CSSValue represents a list of values and also implements the CSSValueList interface.

See Also

CSSPrimitiveValue, CSSValueList

Returned by: CSSStyleDeclaration.getPropertyCSSValue(), CSSValueList.item()

CSSValueList

DOM Level 2 CSS

a CSSValue that holds an array of CSSValue objects

CSSValue → CSSValueList

Properties

`readonly unsigned long length`
: The number of CSSValue objects in this array.

Methods

`item( )`
: Returns the CSSValue object at the specified position in the array, or `null` if the specified position is negative or if it is greater than or equal to `length`.

Description

This interface represents an array of CSSValue objects and is itself a type of CSSValue. The `item( )` method can be used to retrieve the CSSValue object at a specified position, but in JavaScript, it is easier to simply index the array using standard square-bracket notation.

The order of CSSValue objects in a CSSValueList array is the order in which they appear in the CSS style declaration. Some CSS attributes whose value is a CSSValueList may also have the value `none`. This special value translates into a CSSValueList object with a `length` of 0.

CSSValueList.item( )

DOM Level 2 CSS

get the CSSValue at the specified position

Synopsis

```
CSSValue item(unsigned long index);
```

Arguments

index
: The position of the desired CSSValue.

Returns

The CSSValue object at the specified position in this CSSValueList, or `null` if *index* is negative or is greater than or equal to `length`.

Document

DOM Level 1 Core

an HTML or XML document

Node → Document

Subinterfaces

HTMLDocument

Also Implements

DocumentCSS
: If the implementation supports the CSS module, the object that implements this Document interface also implements the DocumentCSS interface and its `getOverrideStyle( )` method.

DocumentEvent
: If the implementation supports the Events module, the object that implements this Document interface also implements the DocumentEvent interface and its `createEvent( )` method.

DocumentRange
: If the implementation supports the Range module, the object that implements this Document interface also implements the DocumentRange interface and its `createRange( )` method.

DocumentStyle
: If the implementation supports the StyleSheets module, the object that implements this Document interface also implements the DocumentStyle interface and its `styleSheets` property.

DocumentTraversal
: If the implementation supports the Traversal module, the object that implements this Document interface also implements the DocumentTraversal interface and its `createNodeIterator( )` and `createTreeWalker( )` methods.

DocumentView
: If the implementation supports the Views module, the object that implements this Document interface also implements the DocumentView interface and its `defaultView` property.

Because these interfaces define commonly implemented additions to the Document interface, their properties and methods are listed and documented here, as if they were directly part of the Document interface.

Properties

`readonly AbstractView defaultView` *[DOM Level 2 Views]*
: The default view of this document. In a web-browser environment, this property specifies the Window object (which implements the AbstractView interface) in which the document is displayed.

  Note that this property is technically part of the DocumentView interface; it is defined by the Document object only in implementations that support the Views module.

`readonly DocumentType doctype`
: For XML documents with a `<!DOCTYPE>` declaration, specifies a DocumentType node that represents the document's DTD. For HTML documents and for XML documents with no `<!DOCTYPE>`, this property is `null`. Note that the property is read-only, and the node to which it refers is also read-only.

`readonly Element documentElement`
: A reference to the root element of the document. For HTML documents, this property is always the Element object representing the `<html>` tag. This root element is also available through the `childNodes[]` array inherited from Node.

`readonly DOMImplementation implementation`
: The DOMImplementation object that represents the implementation that created this document.

`readonly StyleSheetList styleSheets` *[DOM Level 2 StyleSheets]*
: A collection of objects representing all style sheets embedded in or linked into a document. In HTML documents, this includes style sheets defined with `<link>` and `<style>` tags.

  Note that this property is technically part of the DocumentStyle interface; it is defined by the Document object only in implementations that support the StyleSheets module.

Methods

`createAttribute( )`
: Creates a new Attr node with the specified name.

`createAttributeNS( )` *[DOM Level 2]*
: Creates a new Attr node with the specified name and namespace.

`createCDATASection( )`
: Creates a new CDATASection node containing the specified text.

`createComment( )`
: Creates a new Comment node containing the specified string.

`createDocumentFragment( )`
: Creates a new, empty DocumentFragment node.

`createElement( )`
: Creates a new Element node with the specified tag name.

`createElementNS( )` *[DOM Level 2]*
: Creates a new Element node with the specified tag name and namespace.

`createEntityReference( )`
: Creates a new EntityReference node that refers to an entity with the specified name. If the DocumentType object for this document defines an Entity with that name, the newly created EntityReference node is given the same read-only children that the Entity node has.

`createEvent( )` *[DOM Level 2 Events]*
: Creates a new synthetic Event object of the named type. Technically, this method is defined by the DocumentEvent interface; it is implemented by the Document object only in implementations that support the Events module.

`createNodeIterator( )` *[DOM Level 2 Traversal]*
: Creates a NodeIterator object. This method is technically part of the DocumentTraversal interface; it is implemented by the Document object only in implementations that support the Traversal module.

`createProcessingInstruction( )`
: Creates a new ProcessingInstruction node with the specified target and data string.

`createRange( )` *[DOM Level 2 Range]*
: Creates a new Range object. This method is technically part of the DocumentRange interface; it is implemented by the Document object only in implementations that support the Range module.

`createTextNode( )`
: Creates a new Text node to represent the specified text.

`createTreeWalker( )` *[DOM Level 2 Traversal]*
: Creates a TreeWalker object. This method is technically part of the DocumentTraversal interface; it is implemented by the Document object only in implementations that support the Traversal module.

`getElementById( )` *[DOM Level 2]*
: Returns a descendant Element of this document that has the specified value for its `id` attribute, or `null` if no such Element exists in the document.

`getElementsByTagName( )`
: Returns an array (technically a NodeList) of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the order in which they appear in the document source.

`getElementsByTagNameNS( )` *[DOM Level 2]*
: Returns an array of all Element nodes that have the specified tag name and namespace.

`getOverrideStyle( )` *[DOM Level 2 CSS]*
: Gets the CSS override style information for the specified Element (and an optional named pseudoelement). This method is technically part of the DocumentCSS interface; it is implemented by the Document object only in implementations that support the CSS module.

`importNode( )` *[DOM Level 2]*
: Makes a copy of a node from some other document that is suitable for insertion into this document.

Description

The Document interface is the root node of a document tree. A Document node may have multiple children, but only one of those children may be an Element node: it is the root

element of the document. The root element is most easily accessed through the `documentElement` property. The `doctype` and `implementation` properties provide access to the DocumentType object (if any) and the DOMImplementation object for this document.

Most of the methods defined by the Document interface are "factory methods" that are used to create various types of nodes that can be inserted into this document. The notable exceptions are `getElementById( )` and `getElementsByTagName( )`, which are quite useful for finding a specific Element or a set of related Element nodes within the document tree.

Contrast this Document object to the Document object documented in the client-side reference section of this book. The Level 0 properties and methods of that client-side Document object are formally defined by the DOM standard in the HTMLDocument interface. See "HTMLDocument" in this reference section for the DOM equivalent of the traditional client-side JavaScript Document object.

The Document interface is defined by the Core module of the DOM Level 2 specification. A number of the other modules define "add-on" interfaces that are intended to be implemented by the same object that implements the Document interface. For example, if an implementation supports the CSS module, the object that implements this interface also implements the DocumentCSS interface. In JavaScript, the properties and methods of these add-on interfaces can be used as if they were defined by Document, and for that reason, those methods and properties are listed here. See the earlier "Also Implements" section for a full list of the add-on interfaces for Document.

See Also

HTMLDocument

Type of: AbstractView.document, HTMLFrameElement.contentDocument, HTMLIFrameElement.contentDocument, HTMLObjectElement.contentDocument, Node.ownerDocument

Returned by: DOMImplementation.createDocument( )

Document.createAttribute( )

DOM Level 1 Core

create a new Attr node

Synopsis

```
Attr createAttribute(String name)
    throws DOMException;
```

Arguments

name

The name for the newly created attribute.

Returns

A newly created Attr node with its `nodeName` property set to *name*.

Throws

This method throws a DOMException with a `code` of `INVALID_CHARACTER_ERR` if *name* contains an illegal character.

See Also

Attr, Element.setAttribute( ), Element.setAttributeNode( )

Document.createAttributeNS( ) DOM Level 2 Core

create an Attr with a name and namespace

Synopsis

```
Attr createAttributeNS(String namespaceURI,
                       String qualifiedName)
    throws DOMException;
```

Arguments

namespaceURI
: The unique identifier of the namespace for the Attr, or `null` for no namespace.

qualifiedName
: The qualified name of the attribute, which should include a namespace prefix, a colon, and a local name.

Returns

A newly created Attr node with the specified name and namespace.

Throws

This method may throw a DOMException with one of the following `code` values in the following circumstances:

`INVALID_CHARACTER_ERR`
: *qualifiedName* contains an illegal character.

`NAMESPACE_ERR`
: *qualifiedName* is malformed, or there is a mismatch between *qualifiedName* and *namespaceURI*.

`NOT_SUPPORTED_ERR`
: The implementation does not support XML documents and therefore does not implement this method.

Description

`createAttributeNS( )` is just like `createAttribute( )` except that the created Attr node has a name and namespace instead of just a name. This method is useful only with XML documents that use namespaces.

Document.createCDATASection( ) DOM Level 1 Core

create a new CDATASection node

Synopsis

```
CDATASection createCDATASection(String data)
    throws DOMException;
```

Arguments

data
: The text of the CDATASection to create.

Returns

A newly created CDATASection node, with the specified *data* as its contents.

Throws

If the document is an HTML document, this method throws a DOMException with a code of `NOT_SUPPORTED_ERR` because HTML documents do not allow CDATASection nodes.

Document.createComment( )

DOM Level 1 Core

create a new Comment node

Synopsis

```
Comment createComment(String data);
```

Arguments

data
: The text of the Comment node to create.

Returns

A newly created Comment node, with the specified *data* as its text.

Document.createDocumentFragment( )

DOM Level 1 Core

create a new, empty DocumentFragment node

Synopsis

```
DocumentFragment createDocumentFragment( );
```

Returns

A newly created DocumentFragment node with no children.

Document.createElement( )

DOM Level 1 Core

create a new Element node

Synopsis

```
Element createElement(String tagName)
    throws DOMException;
```

Arguments

tagName
: The tag name of the Element to be created. Since HTML tags are case-insensitive, you may use any capitalization for HTML tag names. XML tag names are case-sensitive.

Returns

A newly created Element node with the specified tag name.

Throws

This method throws a DOMException with a code of INVALID_CHARACTER_ERR if *tagName* contains an illegal character.

Document.createElementNS( )

DOM Level 2 Core

create a new Element node using a namespace

Synopsis

```
Element createElementNS(String namespaceURI,
                        String qualifiedName)
    throws DOMException;
```

Arguments

namespaceURI
: The unique identifier for the namespace of the new Element, or null for no namespace.

qualifiedName
: The qualified name of the new Element. This should include a namespace prefix, a colon, and a local name.

Returns

A newly created Element node, with the specified tag name and namespace.

Throws

This method may throw a DOMException with one of the following code values in the following circumstances:

INVALID_CHARACTER_ERR
: *qualifiedName* contains an illegal character.

NAMESPACE_ERR
: *qualifiedName* is malformed, or there is a mismatch between *qualifiedName* and *namespaceURI*.

NOT_SUPPORTED_ERR
: The implementation does not support XML documents and therefore does not implement this method.

Description

createElementNS( ) is just like createElement( ) except that the created Element node has a name and namespace instead of just a name. This method is useful only with XML documents that use namespaces.

Document.createEntityReference( )

DOM Level 1 Core

create a new EntityReference node

Synopsis

```
EntityReference createEntityReference(String name)
    throws DOMException;
```

Arguments

name
: The name of the referenced entity.

Returns

A new EntityReference node that references an entity with the specified name.

Throws

This method may throw a DOMException with one of the following code values:

`INVALID_CHARACTER_ERR`
: The specified entity name contains an illegal character.

`NOT_SUPPORTED_ERR`
: This is an HTML document and does not support entity references.

Description

This method creates and returns an EntityReference node that refers to an entity with the specified name. Note that it always throws an exception if this is an HTML document, because HTML does not allow entity references. If this document has a DocumentType node, and if that DocumentType defines an Entity object with the specified name, the returned EntityReference has the same children as the referenced Entity node.

Document.createEvent( )

DOM Level 2 Events

create an Event object

Synopsis

```
Event createEvent(String eventType)
    throws DOMException;
```

Arguments

eventType
: The name of the event module for which an Event object is desired. See the "Description" section for a list of valid event types.

Returns

A newly created Event object of the specified type.

Throws

This method throws a DOMException with a code of NOT_SUPPORTED_ERR if the implementation does not support events of the requested type.

Description

This method creates a new event type of the type specified by the *eventType* argument. Note that the value of this argument should not be the (singular) name of the event interface to be created, but instead should be the (plural) name of the DOM module that defines that interface. The following table shows the legal values for *eventType* and the event interface each value creates.

eventType argument	Event interface	Initialization method
HTMLEvents	Event	initEvent( )
MouseEvents	MouseEvent	initMouseEvent( )
UIEvents	UIEvent	initUIEvent( )
MutationEvents	MutationEvent	initMutationEvent( )

After creating an Event object with this method, you must initialize the object with the initialization method shown in the table. See the appropriate Event interface reference page for details about the initialization method.

This method is actually defined not by the Document interface but by the DocumentEvent interface. If an implementation supports the Events module, the Document object always implements the DocumentEvent interface and supports this method.

See Also

Event, MouseEvent, MutationEvent, UIEvent

Document.createNodeIterator( )

DOM Level 2 Traversal

create a NodeIterator for this document

Synopsis

```
NodeIterator createNodeIterator(Node root,
                                unsigned long whatToShow,
                                NodeFilter filter,
                                boolean entityReferenceExpansion)
    throws DOMException;
```

Arguments

root
: The root of the subtree over which the NodeIterator is to iterate.

whatToShow
: A bitmask of one or more NodeFilter flags that specify which types of nodes should be returned by this NodeIterator.

filter
: An optional node filter function for the NodeIterator, or null for no node filter.

entityReferenceExpansion
: `true` if the NodeIterator should expand entity references in XML documents, or `false` otherwise.

Returns

A newly created NodeIterator object.

Throws

This method throws a DOMException with a `code` of `NOT_SUPPORTED_ERR` if the *root* argument is `null`.

Description

This method creates and returns a new NodeIterator object to iterate over the subtree rooted at the *root* node, using the specified filters.

This method is not actually part of the Document interface but is instead defined by the DocumentTraversal interface. If an implementation supports the Traversal module, the Document object always implements DocumentTraversal and defines this method.

See Also

Document.createTreeWalker( ), NodeFilter, NodeIterator

Document.createProcessingInstruction( )

DOM Level 1 Core

create a ProcessingInstruction node

Synopsis

```
ProcessingInstruction createProcessingInstruction(String target,
                                                   String data)
    throws DOMException;
```

Arguments

target
: The target of the processing instruction.

data
: The content text of the processing instruction.

Returns

A newly created ProcessingInstruction node.

Throws

This method may throw a DOMException with one of the following `code` values in the following circumstances:

`INVALID_CHARACTER_ERR`
: The specified *target* contains an illegal character.

`NOT_SUPPORTED_ERR`
: This is an HTML document and does not support processing instructions.

Document.createRange( )

DOM Level 2 Range

create a Range object

Synopsis

```
Range createRange( );
```

Returns

A newly created Range object with both boundary points set to the beginning of the document.

Description

This method creates a Range object that can be used to represent a region of this document or of a DocumentFragment associated with this document.

Note that this method is actually defined not by the Document interface but by the DocumentRange interface. If an implementation supports the Range module, the Document object always implements DocumentRange and defines this method.

Document.createTextNode( )

DOM Level 1 Core

create a new Text node

Synopsis

```
Text createTextNode(String data);
```

Arguments

data
: The content of the Text node.

Returns

A newly created Text node that represents the specified *data* string.

Document.createTreeWalker( )

DOM Level 2 Traversal

create a TreeWalker for this document

Synopsis

```
TreeWalker createTreeWalker(Node root,
                            unsigned long whatToShow,
                            NodeFilter filter,
                            boolean entityReferenceExpansion)
    throws DOMException;
```

Arguments

root
: The root of the subtree over which this TreeWalker is to walk.

whatToShow
: A bitmask of one or more NodeFilter flags that specify which types of nodes should be returned by this TreeWalker.

filter
: An optional node filter function for the TreeWalker, or `null` for no node filter.

entityReferenceExpansion
: `true` if the TreeWalker should expand entity references in XML documents, or `false` otherwise.

Returns

A newly created TreeWalker object.

Throws

This method throws a DOMException with a `code` of `NOT_SUPPORTED_ERR` if the *root* argument is `null`.

Description

This method creates and returns a new TreeWalker object to traverse the subtree rooted at the *root* node, using the specified filters.

This method is not actually part of the Document interface but is instead defined by the DocumentTraversal interface. If an implementation supports the Traversal module, the Document object always implements DocumentTraversal and defines this method.

See Also

Document.createNodeIterator( ), NodeFilter, TreeWalker

Document.getElementById( ) DOM Level 2 Core; in DOM Level 1, defined by HTMLDocument

find an element with the specified unique ID

Synopsis

```
Element getElementById(String elementId);
```

Arguments

elementId
: The value of the `id` attribute of the desired element.

Returns

The Element node that represents the document element with the specified `id` attribute, or `null` if no such element is found.

Description

This method searches the document for an Element node with an `id` attribute whose value is *elementId*, and returns that Element. If no such Element is found, it returns `null`. The

value of the id attribute is intended to be unique within a document, and if this method finds more than one Element with the specified *elementId*, it may return one at random or it may return null.

In HTML documents, this method always searches for attributes named id. In XML documents, however, it searches for any attribute whose *type* is id, regardless of what the attribute name is. If XML attribute types are not known (because, for example, the XML parser could not locate the document's DTD), this method always returns null.

This is an important and commonly used method since it provides a simple way to obtain the Element object that represents a specific document element. Note that it provides functionality similar to the nonstandard document.all[] array defined by Internet Explorer 4 and later. Finally, note that the name of this method ends with "Id", not with "ID"; be careful not to misspell it.

See Also

Document.getElementsByTagName(), Element.getElementsByTagName(), HTMLDocument.getElementsByName()

Document.getElementsByTagName( )

DOM Level 1 Core

return all Element nodes with the specified name

Synopsis

```
Node[] getElementsByTagName(String tagname);
```

Arguments

tagname

The tag name of the Element nodes to be returned, or the wildcard string "*" to return all Element nodes in the document regardless of tag name. For HTML documents, tag names are compared in a case-insensitive fashion.

Returns

A read-only array (technically, a NodeList) of all Element nodes in the document tree with the specified tag name. The returned Element nodes are in the same order in which they appear in the document source.

Description

This method returns a NodeList (which you can treat as a read-only array) that contains all Element nodes from the document that have the specified tag name, in the order in which they appear in the document source. The NodeList is "live"; i.e., its contents are automatically updated as necessary if elements with the specified tag name are added to or removed from the document.

HTML documents are case-insensitive, and you can specify *tagname* using any capitalization; it matches all tags with the same name in the document, regardless of how those tags are capitalized in the document source. XML documents, on the other hand, are case-sensitive, and *tagname* matches only tags with the same name and exactly the same capitalization in the document source.

Note that the Element interface defines a method by the same name that searches only a subtree of the document. Also, the HTMLDocument interface defines `getElementsByName( )`, which searches for elements based on the value of their `name` attributes rather than their tag names.

Example

You can find and iterate through all `<h1>` tags in a document with code like the following:

```
var headings = document.getElementsByTagName("h1");
for(var i = 0; i < headings.length; i++) {  // Loop through the returned tags
    var h = headings[i];
    // Now do something with the <h1> element in the h variable
}
```

See Also

Document.getElementById(), Element.getElementsByTagName(), HTMLDocument.getElementsByName()

Document.getElementsByTagNameNS( )

DOM Level 2 Core

return all Element nodes with a specified name and namespace

Synopsis

```
Node[] getElementsByTagNameNS(String namespaceURI,
                              String localName);
```

Arguments

namespaceURI
: The unique identifier of the namespace of the desired elements, or "*" to match all namespaces.

localName
: The local name of the desired elements, or "*" to match any local name.

Returns

A read-only array (technically, a NodeList) of all Element nodes in the document tree that have the specified namespace and local name.

Description

This method works just like `getElementsByTagName( )` except that it searches for elements by namespace and name. It is useful only with XML documents that use namespaces.

Document.getOverrideStyle( )

DOM Level 2 CSS

get the override style for a specified element

Synopsis

```
CSSStyleDeclaration getOverrideStyle(Element elt,
                                     String pseudoElt);
```

Arguments

elt
: The element for which the override style is desired.

pseudoElt
: The pseudoelement of *elt*, or null if there is none.

Returns

A CSSStyleDeclaration object that represents the override style information for the specified element and pseudoelement. The returned object typically also implements the more commonly used CSS2Properties interfaces.

Description

This method returns a CSSStyleDeclaration object (which typically also implements CSS2Properties) for a specified element and optional pseudoelement. You may make use of this returned object to make changes to the displayed style of the specified element without disturbing the inline style of that element and without modifying the style sheets of the document. Conceptually, the returned value represents a style declaration within an "override" style sheet that takes precedence over all other style sheets and inline styles (except for !important declarations in the user style sheet).

Note that this method is defined not by the Document interface but by the DocumentCSS interface. If an implementation supports the CSS module, the Document object always implements DocumentCSS and defines this method.

See Also

CSSStyleDeclaration, CSS2Properties, AbstractView.getComputedStyle( ), HTMLElement.style

Document.importNode( )

DOM Level 2 Core

copy a node from another document for use in this document

Synopsis

```
Node importNode(Node importedNode,
                boolean deep)
    throws DOMException;
```

Arguments

importedNode
: The node to be imported.

deep
: If true, recursively copy all descendants of *importedNode* as well.

Returns

A copy of *importedNode* (and possibly all of its descendants) with its ownerDocument set to this document.

Throws

This method throws a DOMException with a code of NOT_SUPPORTED_ERR if *importedNode* is a Document or DocumentType node, since those types of nodes cannot be imported.

Description

This method is passed a node defined in another document and returns a copy of the node that is suitable for insertion into this document. If *deep* is true, all descendants of the node are also copied. The original node and its descendants are not modified in any way. The returned copy has its ownerDocument property set to this document but has a parentNode of null since it has not yet been inserted into the document. EventListener functions registered on the original node or tree are not copied.

When an Element node is imported, only the attributes that are explicitly specified in the source document are imported with it. When an Attr node is imported, its specified property is automatically set to true.

See Also

Node.cloneNode( )

DocumentCSS

see Document

DocumentEvent

see Document

DocumentFragment

DOM Level 1 Core

adjacent nodes and their subtrees

Node → DocumentFragment

Description

The DocumentFragment interface represents a portion—or fragment—of a document. More specifically, it represents one or more adjacent Document nodes and all of the descendants of each. DocumentFragment nodes are never part of a document tree, and the inherited parentNode property is always null. DocumentFragment nodes exhibit a special behavior that makes them quite useful, however: when a request is made to insert a DocumentFragment into a document tree, it is not the DocumentFragment node itself that is inserted but each of the children of the DocumentFragment instead. This makes DocumentFragment useful as a temporary placeholder for nodes that you wish to insert, all at once, into a document. DocumentFragment is also particularly useful for implementing document cut, copy, and paste operations, particularly when combined with the Range interface.

You can create a new, empty DocumentFragment with Document.createDocumentFragment( ), or you can use Range.extractContents( ) or Range.cloneContents( ) to obtain a DocumentFragment that contains a fragment of an existing document.

See Also

Range

Returned by: Document.createDocumentFragment( ), Range.cloneContents( ), Range.extractContents( )

DocumentRange

see Document

DocumentStyle

see Document

DocumentTraversal

see Document

DocumentType

DOM Level 1 XML

the DTD of an XML document

Node → DocumentType

Properties

`readonly NamedNodeMap entities`
: This NamedNodeMap is a list of Entity objects declared in the DTD and allows Entity objects to be queried by name. Note that XML parameter entities are not included. This NamedNodeMap is immutable—its contents may not be altered.

`readonly String internalSubset` *[DOM Level 2]*
: The internal subset of the DTD (i.e., the portion of the DTD that appears in the document itself rather than in an external file). The delimiting square brackets of the internal subset are not part of the returned value. If there is no internal subset, this property is `null`.

`readonly String name`
: The name of the document type. This is the identifier that immediately follows `<!DOCTYPE>` at the start of an XML document, and it is the same as the tag name of the document's root element.

`readonly NamedNodeMap notations`
: A NamedNodeMap that contains Notation objects representing all notations declared in the DTD. It also allows Notation objects to be looked up by notation name. This NamedNodeMap is immutable—its contents may not be altered.

`readonly String publicId` *[DOM Level 2]*
: The public identifier of the external subset of the DTD, or `null` if none was specified.

`readonly String systemId` *[DOM Level 2]*
: The system identifier of the external subset of the DTD, or `null` if none was specified.

Description

This infrequently used interface represents the DTD of an XML document. Programmers working exclusively with HTML documents never need to use this interface.

Because a DTD is not part of a document's content, DocumentType nodes never appear in the document tree. If an XML document has a DTD, the DocumentType node for that DTD is available through the `doctype` property of the Document node.

DocumentType nodes are immutable and may not be modified in any way.

See Also

Document, Entity, Notation

Type of: Document.doctype

Passed to: DOMImplementation.createDocument( )

Returned by: DOMImplementation.createDocumentType( )

DocumentView

see Document

DOMException

DOM Level 1 Core

signals exceptions or errors for core DOM objects

Constants

The following constants define the legal values for the `code` property of a DOMException object. Note that these constants are static properties of DOMException, not properties of individual exception objects.

`unsigned short INDEX_SIZE_ERR = 1`
: Indicates an out-of-bounds error for an array or string index.

`unsigned short DOMSTRING_SIZE_ERR = 2`
: Indicates that a requested text is too big to fit into a string in the current JavaScript implementation.

`unsigned short HIERARCHY_REQUEST_ERR = 3`
: Indicates that an attempt was made to place a node somewhere illegal in the document tree hierarchy.

`unsigned short WRONG_DOCUMENT_ERR = 4`
: Indicates an attempt to use a node with a document that is different from the document that created the node.

`unsigned short INVALID_CHARACTER_ERR = 5`
: Indicates that an illegal character is used (in an element name, for example).

`unsigned short NO_DATA_ALLOWED_ERR = 6`
: Not currently used.

`unsigned short NO_MODIFICATION_ALLOWED_ERR = 7`
: Indicates that an attempt was made to modify a node that is read-only and does not allow modifications. Entity, EntityReference, and Notation nodes, and all of their descendants, are read-only.

`unsigned short NOT_FOUND_ERR = 8`
: Indicates that a node was not found where it was expected.

`unsigned short NOT_SUPPORTED_ERR = 9`
: Indicates that a method or property is not supported in the current DOM implementation.

`unsigned short INUSE_ATTRIBUTE_ERR = 10`
: Indicates that an attempt was made to associate an Attr with an Element when that Attr node was already associated with a different Element node.

`unsigned short INVALID_STATE_ERR = 11` *[DOM Level 2]*
: Indicates an attempt to use an object that is not yet, or is no longer, in a state that allows such use.

`unsigned short SYNTAX_ERR = 12` *[DOM Level 2]*
: Indicates that a specified string contains a syntax error. Commonly used with CSS property specifications.

`unsigned short INVALID_MODIFICATION_ERR = 13` *[DOM Level 2]*
: Indicates an attempt to modify the type of a CSSRule or CSSValue object.

`unsigned short NAMESPACE_ERR = 14` *[DOM Level 2]*
: Indicates an error involving element or attribute namespaces.

`unsigned short INVALID_ACCESS_ERR = 15` *[DOM Level 2]*
: Indicates an attempt to access an object in a way that is not supported by the implementation.

Properties

`unsigned short code`
: An error code that provides some detail about what caused the exception. The legal values (and their meanings) for this property are defined by the constants just listed.

Description

A DOMException object is thrown when a DOM method or property is used incorrectly or in an inappropriate context. The value of the `code` property indicates the general type of exception that occurred. Note that a DOMException may be thrown when reading or writing a property of an object as well as when calling a method of an object.

The descriptions of object properties and methods in this reference include a list of exception types they may throw. Note, however, that certain commonly thrown exceptions are omitted from these lists. A DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` is thrown any time an attempt is made to modify a read-only node, such as an Entity node or one of its descendants. Thus, most methods and read/write properties of the Node interface (and of its subinterfaces) may throw this exception. Because read-only nodes appear only in XML documents and not in HTML documents, and because it applies so universally to the methods and writable properties of Node objects, the `NO_MODIFICATION_ALLOWED_ERR` exception is omitted from the descriptions of those methods and properties.

Similarly, many DOM methods and properties that return strings may throw a DOMException with a `code` of `DOMSTRING_SIZE_ERR`, which indicates that the text to be returned is too long to be represented as a string value in the underlying JavaScript implementation. Although this type of exception may theoretically be thrown by many properties and methods, it is very rare in practice and is omitted from the descriptions of those methods and properties.

Note that not all exceptions in the DOM are signaled with a DOMException. Exceptions having to do with events and event handling cause an EventException object to be thrown, and exceptions involving the DOM Range module cause a RangeException to be thrown.

See Also

EventException, RangeException

DOMImplementation

DOM Level 1 Core

methods independent of any particular document

Also Implements

DOMImplementationCSS, HTMLDOMImplementation

If a DOM implementation supports the HTML and CSS modules, the DOMImplementation object also implements the DOMImplementationCSS and HTMLDOMImplementation interfaces. For convenience, the methods of these trivial interfaces are listed here along with the core DOMImplementation methods.

Methods

`createCSSStyleSheet( )` *[DOM Level 2 CSS]*

This DOMImplementationCSS method creates a new CSSStyleSheet object.

`createDocument( )` *[DOM Level 2]*

Creates a new Document object with a root element (the `documentElement` property of the returned Document object) of the specified type.

`createDocumentType( )` *[DOM Level 2]*

Creates a new DocumentType node.

`createHTMLDocument( )` *[DOM Level 2 HTML]*

This HTMLDOMImplementation method creates a new HTMLDocument object and populates it with `<html>`, `<head>`, `<title>`, and `<body>` elements.

`hasFeature( )`

Checks whether the current implementation supports a specified version of a named feature.

Description

The DOMImplementation interface and its HTMLDOMImplementation and DOMImplementationCSS subinterfaces are placeholders for methods that are not specific to any particular Document object but rather are "global" to an implementation of the DOM. You can obtain a reference to the DOMImplementation object through the `implementation` property of any Document object.

See Also

Type of: Document.implementation

DOMImplementation.createCSSStyleSheet( ) DOM Level 2 CSS

create a CSSStyleSheet object

Synopsis

```
CSSStyleSheet createCSSStyleSheet(String title,
                                  String media)
    throws DOMException;
```

Arguments

title
: The title of the style sheet.

media
: A comma-separated list of media types to which the style sheet should apply.

Returns

A CSSStyleSheet object.

Throws

A DOMException with a code of `SYNTAX_ERR` if the *media* argument is malformed.

Description

This method creates a new CSSStyleSheet object. Note, however, that as of Level 2, the DOM standard does not yet define any way to associate a newly created CSSStyleSheet object with a document.

`createCSSStyleSheet( )` is defined not by the DOMImplementation interface but by its DOMImplementationCSS subinterface. If an implementation supports the "CSS" feature, its DOMImplementation object implements this method.

DOMImplementation.createDocument( ) DOM Level 2 Core

create a new Document and the specified root element

Synopsis

```
Document createDocument(String namespaceURI,
                        String qualifiedName,
                        DocumentType doctype)
    throws DOMException;
```

Arguments

namespaceURI
: The unique identifier of the namespace of the root element to be created for the document, or `null` for no namespace.

qualifiedName
: The name of the root element to be created for this document. If *namespaceURI* is not `null`, this name should include a namespace prefix and a colon.

doctype
: The DocumentType object for the newly created Document, or `null` if none is desired.

Returns

A Document object with its `documentElement` property set to a root Element node of the specified type.

Throws

This method may throw a DOMException with the following `code` values in the following circumstances:

`INVALID_CHARACTER_ERR`
: *qualifiedName* contains an illegal character.

`NAMESPACE_ERR`
: *qualifiedName* is malformed, or there is a mismatch between *qualifiedName* and *namespaceURI*.

`NOT_SUPPORTED_ERR`
: The current implementation does not support XML documents and has not implemented this method.

`WRONG_DOCUMENT_ERR`
: *doctype* is already in use for another document or was created by a different DOMImplementation object.

Description

This method creates a new Document object and the specified root `documentElement` object for that document. If the *doctype* argument is non-`null`, the `ownerDocument` property of this DocumentType object is set to the newly created document.

This method is used to create XML documents and may not be supported by HTML-only implementations. Use `createHTMLDocument( )` to create a new HTML document.

See Also

createDocumentType( ), createHTMLDocument( )

DOMImplementation.createDocumentType( )

DOM Level 2 Core

create a DocumentType node

Synopsis

```
DocumentType createDocumentType(String qualifiedName,
                                String publicId,
                                String systemId)
    throws DOMException;
```

Arguments

qualifiedName
: The name of the document type. If you are using XML namespaces, this may be a qualified name that specifies a namespace prefix and a local name separated by a colon.

publicId
: The public identifier of the document type, or `null`.

systemId
: The system identifier of the document type, or `null`. This argument typically specifies the local filename of a DTD file.

Returns

A new `DocumentType` object with an `ownerDocument` property of `null`.

Throws

This method may throw a DOMException with one of the following `code` values:

`INVALID_CHARACTER_ERR`
: *qualifiedName* contains an illegal character.

`NAMESPACE_ERR`
: *qualifiedName* is malformed.

`NOT_SUPPORTED_ERR`
: The current implementation does not support XML documents and has not implemented this method.

Description

This method creates a new DocumentType node. This method specifies only an external subset of the document type. As of Level 2, the DOM standard does not provide any way for specifying an internal subset, and the returned DocumentType does not define any Entity or Notation nodes. This method is useful only with XML documents and may not be supported by HTML-only implementations.

DOMImplementation.createHTMLDocument( ) DOM Level 2 HTML

create a skeletal HTML document

Synopsis

```
HTMLDocument createHTMLDocument(String title);
```

Arguments

title
: The title of the document. This text is used as the content of the `<title>` element of the newly created document.

Returns

The new HTMLDocument object.

Description

This method creates a new HTMLDocument object with a skeletal document tree that includes the specified title. The `documentElement` property of the returned object is an `<html>` element, and this root element has `<head>` and `<body>` tags as its children. The `<head>` element in turn has a `<title>` child, which has the specified *`title`* string as its child.

`createHTMLDocument( )` is defined not by the DOMImplementation interface but by its HTMLDOMImplementation subinterface. If an implementation supports the "HTML" feature, its DOMImplementation object implements this method.

See Also

DOMImplementation.createDocument( )

DOMImplementation.hasFeature( )

DOM Level 1 Core

determine whether the implementation supports a feature

Synopsis

```
boolean hasFeature(String feature,
                   String version);
```

Arguments

`feature`
: The name of the feature for which support is being tested. The set of valid feature names for the DOM Level 2 standard is listed in the upcoming table. Feature names are case-insensitive.

`version`
: The feature version number for which support is being tested, or `null` or the empty string `""` if support for any version of the feature is sufficient. In the Level 2 DOM specification, supported version numbers are 1.0 and 2.0.

Returns

`true` if the implementation completely supports the specified version of the specified feature, or `false` otherwise. If no version number is specified, the method returns `true` if the implementation completely supports any version of the specified feature.

Description

The W3C DOM standard is modular, and implementations are not required to implement all modules or features of the standard. This method is used to test whether a DOM implementation supports a named module of the DOM specification. The availability information for each entry in this DOM reference includes the name of the module. Note that although Internet Explorer 5 and 5.5 include partial support for the DOM Level 1 specification, this important method is not supported before IE 6.

The complete set of module names that may be used as the *`feature`* argument are shown in the following table.

Feature	Description
Core	Node, Element, Document, Text, and the other fundamental interfaces required by all DOM implementations are implemented. All conforming implementations must support this module.
HTML	HTMLElement, HTMLDocument, and the other HTML-specific interfaces are implemented.
XML	Entity, EntityReference, ProcessingInstruction, Notation, and the other node types that are useful only with XML documents are implemented.
StyleSheets	Simple interfaces describing generic style sheets are implemented.
CSS	Interfaces that are specific to CSS style sheets are implemented.
CSS2	The CSS2Properties interface is implemented.
Events	The basic event-handling interfaces are implemented.
UIEvents	The interfaces for user-interface events are implemented.
MouseEvents	The interfaces for mouse events are implemented.
HTMLEvents	The interfaces for HTML events are implemented.
MutationEvents	The interfaces for document mutation events are implemented.
Range	The interfaces for manipulating ranges of a document are implemented.
Traversal	The interfaces for advanced document traversal are implemented.
Views	The interfaces for document views are implemented.

Example

You might use this method in code like the following:

```
// Check whether the browser supports the DOM Level 2 Traversal API
if (document.implementation &&
    document.implementation.hasFeature &&
    document.implementation.hasFeature("Traversal", "2.0")) {
  // If so, use it here...
}
else {
  // If not, traverse the document some other way
}
```

See Also

Node.isSupported( )

DOMImplementationCSS

see DOMImplementation

Element

DOM Level 1 Core

an HTML or XML element

Node → Element

Subinterfaces

HTMLElement

Properties

`readonly String tagName`
: The tag name of the element. This is the string "P" for an HTML <p> element, for example. For HTML documents, the tag name is returned in uppercase, regardless of its capitalization in the document source. XML documents are case-sensitive, and the tag name is returned exactly as it is written in the document source. This property has the same value as the `nodeName` property of the Node interface.

Methods

`getAttribute( )`
: Returns the value of a named attribute as a string.

`getAttributeNS( )` *[DOM Level 2]*
: Returns the string value of an attribute specified by local name and namespace URI. Useful only with XML documents that use namespaces.

`getAttributeNode( )`
: Returns the value of a named attribute as an Attr node.

`getAttributeNodeNS( )` *[DOM Level 2]*
: Returns the Attr value of an attribute specified by local name and namespace URI. Useful only with XML documents that use namespaces.

`getElementsByTagName( )`
: Returns an array (technically, a NodeList) of all descendant Element nodes of this element that have the specified tag name, in the order in which they appear in the document.

`getElementsByTagNameNS( )` *[DOM Level 2]*
: Like `getElementsByTagName( )`, except that the element tag name is specified by local name and namespace URI. Useful only with XML documents that use namespaces.

`hasAttribute( )` *[DOM Level 2]*
: Returns `true` if this element has an attribute with the specified name, or `false` otherwise. Note that this method returns `true` if the named attribute is explicitly specified in the document source or if the document's DTD specifies a default value for the named attribute.

`hasAttributeNS( )` *[DOM Level 2]*
: Like `hasAttribute( )`, except that the attribute is specified by a combination of local name and namespace URI. This method is useful only with XML documents that use namespaces.

`removeAttribute( )`
: Deletes the named attribute from this element. Note, however, that this method deletes only attributes that are explicitly specified in the document source for this element. If the DTD specifies a default value for this attribute, that default becomes the new value of the attribute.

`removeAttributeNS( )` *[DOM Level 2]*
: Like `removeAttribute( )`, except that the attribute to be removed is specified by a combination of local name and namespace URI. Useful only for XML documents that use namespaces.

`removeAttributeNode( )`
: Removes the specified Attr node from the list of attributes for this element. Note that this works only to remove attributes that are explicitly specified in the document source for this attribute. If the DTD specifies a default value for the removed attribute, a new Attr node is created to represent the default value of the attribute.

`setAttribute( )`
: Sets the named attribute to the specified string value. If an attribute with that name does not already exist, a new attribute is added to the element.

`setAttributeNS( )` *[DOM Level 2]*
: Like `setAttribute( )`, except that the attribute to be set is specified by the combination of a local name and a namespace URI. Useful only with XML documents that use namespaces.

`setAttributeNode( )`
: Adds the specified Attr node to the list of attributes for this element. If an attribute with the same name already exists, its value is replaced.

`setAttributeNodeNS( )` *[DOM Level 2]*
: Like `setAttributeNode( )`, but this method is suitable for use with nodes returned by `Document.createAttributeNS( )`. Useful only with XML documents that use namespaces.

Description

The Element interface represents HTML or XML elements or tags. The `tagName` property specifies the name of the element. The `getElementsByTagName( )` method provides a powerful way to locate element descendants of a given element that have a specified tag name. The various other methods of this interface provide access to the attributes of the element. If you give an element a unique identifier using the `id` attribute in your document source, you can then easily locate the Element node that represents that document element with the useful `Document.getElementById( )` method. To create a new Element node for insertion into a document, use `Document.createElement( )`.

In HTML documents (and many XML documents) all attributes have simple string values, and you can use the simple methods `getAttribute( )` and `setAttribute( )` for any attribute manipulation you need to do.

If you are working with XML documents that may contain entity references as part of attribute values, you will have to work with Attr objects and their subtree of nodes. You can get and set the Attr object for an attribute with `getAttributeNode( )` and `setAttributeNode( )`, or you can iterate through the Attr nodes in the `attributes[]` array of the Node interface. If you are working with an XML document that uses XML namespaces, you'll need to use the various methods whose names end with "NS".

In the DOM Level 1 specification, the `normalize( )` method was part of the Element interface. In the Level 2 specification, `normalize( )` is instead part of the Node interface. All Element nodes inherit this method and can still use it.

See Also

HTMLElement, Node

Type of: Attr.ownerElement, Document.documentElement

Passed to: Document.getOverrideStyle( ), AbstractView.getComputedStyle( )

Returned by: Document.createElement( ), Document.createElementNS( ), Document.getElementById( )

Element.getAttribute( )

DOM Level 1 Core

return the string value of a named attribute

Synopsis

```
String getAttribute(String name);
```

Arguments

name
The name of the attribute whose value is to be returned.

Returns

The string value of the named attribute. If the attribute does not have a value specified in the document and does not have a default value specified by the document type, the return value is the empty string ("").

Description

`getAttribute( )` returns the value of a named attribute of an element. In HTML documents, attribute values are always strings, and this method returns the complete attribute value. Note that the objects that represent HTML elements also implement the HTMLElement interface and one of its tag-specific subinterfaces. Therefore, all standard attributes of standard HTML tags are also available directly as properties of the Element object.

In XML documents, attribute values are not available directly as element properties and must be looked up by calling a method. For many XML documents, `getAttribute( )` is a suitable method for doing this. Note, however that in XML attributes may contain entity references, and in order to obtain complete details about such attributes, you must use `getAttributeNode( )` to obtain the Attr node whose subtree represents the complete attribute value. The Attr nodes for an element are also available in an `attributes[]` array inherited from the Node interface. For XML documents that use namespaces, you may need to use `getAttributeNS( )` or `getAttributeNodeNS( )`.

Example

The following code illustrates two different ways of obtaining an attribute value for an HTML `<img>` element:

```
// Get all images in the document
var images = document.body.getElementsByTagName("IMG");
// Get the SRC attribute of the first one
var src0 = images[0].getAttribute("SRC");
// Get the SRC attribute of the second simply by reading the property
var src1 = images[1].src;
```

See Also

Element.getAttributeNode( ), Node.attributes

Element.getAttributeNode( )

DOM Level 1 Core

return the Attr node for the named attribute

Synopsis

```
Attr getAttributeNode(String name);
```

Arguments

name
: The name of the desired attribute.

Returns

An Attr node whose descendants represent the value of the named attribute, or `null` if this element has no such attribute.

Description

`getAttributeNode( )` returns an Attr node that represents the value of a named attribute. Note that this Attr node can also be obtained through the `attributes` property inherited from the Node interface.

The attribute value is represented by the descendants of the Attr nodes. In HTML documents, an Attr node has a single Text node child, and it is always easier to query an attribute value by calling `getAttribute( )`, which returns the value as a string. `getAttributeNode( )` is necessary only when you are working with XML documents that contain entity references in their attribute values.

See Also

Element.getAttribute( ), Element.getAttributeNodeNS( ), Node.attributes

Element.getAttributeNodeNS( )

DOM Level 2 Core

return the Attr node for an attribute with a namespace

Synopsis

```
Attr getAttributeNodeNS(String namespaceURI,
                        String localName);
```

Arguments

namespaceURI
: The URI that uniquely identifies the namespace of this attribute, or `null` for no namespace.

localName
: The identifier that specifies the name of the attribute within its namespace.

Returns

The Attr node whose descendants represent the value of the specified attribute, or `null` if this element has no such attribute.

Description

This method works like `getAttributeNode( )`, except that the attribute is specified by the combination of a namespace URI and a local name defined within that namespace. This method is useful only with XML documents that use namespaces.

See Also

Element.getAttributeNode( ), Element.getAttributeNS( )

Element.getAttributeNS( )

DOM Level 2 Core

get the value of an attribute that uses namespaces

Synopsis

```
String getAttributeNS(String namespaceURI,
                      String localName);
```

Arguments

namespaceURI
: The URI that uniquely identifies the namespace of this attribute, or `null` for no namespace.

localName
: The identifier that specifies the name of the attribute within its namespace.

Returns

The string value of the named attribute. If the attribute is not explicitly specified in the document and does not have a default value specified by the document type, this method returns the empty string.

Description

This method works just like the `getAttribute( )` method, except that the attribute is specified by a combination of namespace URI and local name within that namespace. This method is useful only with XML documents that use namespaces.

See Also

Element.getAttribute( ), Element.getAttributeNodeNS( )

Element.getElementsByTagName( )

DOM Level 1 Core

find descendant elements with a specified tag name

Synopsis

```
Node[] getElementsByTagName(String name);
```

Arguments

name

The tag name of the desired elements, or the value "*" to specify that all descendant elements should be returned, regardless of their tag names.

Returns

An array (technically, a NodeList) of Element objects that are descendants of this element and have the specified tag name.

Description

This method traverses all descendants of this element and returns an array (really a NodeList object) of Element nodes representing all document elements with the specified tag name. The elements in the returned array appear in the same order in which they appear in the source document.

Note that the Document interface also has a `getElementsByTagName( )` method that works just like this one but that traverses the entire document, rather than just the descendants of a single element. Do not confuse this method with `HTMLDocument.getElementsByName( )`, which searches for elements based on the value of their `name` attributes rather than by their tag names.

Example

You can find all `<div>` tags in a document with code like the following:

```
var divisions = document.body.getElementsByTagName("div");
```

And you can find all `<p>` tags within the a `<div>` tag with code like this:

```
var paragraphs = divisions[0].getElementsByTagname("p");
```

See Also

Document.getElementById( ), Document.getElementsByTagName( ), HTMLDocument.getElementsByName( )

Element.getElementsByTagNameNS( )

DOM Level 2 Core

return descendant elements with the specified name and namespace

Synopsis

```
Node[] getElementsByTagNameNS(String namespaceURI,
                              String localName);
```

Arguments

namespaceURI

The URI that uniquely identifies the namespace of the element.

localName

The identifier that specifies the name of the element within its namespace.

Returns

An array (technically, a NodeList) of Element objects that are descendants of this element and have the specified name and namespace.

Description

This method works like `getElementsByTagName( )`, except that the tag name of the desired elements is specified as a combination of a namespace URI and a local name defined within that namespace. This method is useful only with XML documents that use namespaces.

See Also

Document.getElementsByTagNameNS( ), Element.getElementsByTagName( )

Element.hasAttribute( )

DOM Level 2 Core

determine whether this element has a specified attribute

Synopsis

```
boolean hasAttribute(String name);
```

Arguments

name
: The name of the desired attribute.

Returns

`true` if this element has a specified or default value for the named attribute, and `false` otherwise.

Description

This method determines whether an element has an attribute with the specified name, but does not return the value of that attribute. Note that `hasAttribute( )` returns `true` if the named attribute is explicitly specified in the document and also if the named attribute has a default value specified by the document type.

See Also

Attr.specified, Element.getAttribute( ), Element.setAttribute( )

Element.hasAttributeNS( )

DOM Level 2 Core

determine whether this element has a specified attribute

Synopsis

```
boolean hasAttributeNS(String namespaceURI,
                       String localName);
```

Arguments

namespaceURI
: The unique namespace identifier for the attribute, or `null` for no namespace.

localName
: The name of the attribute within the specified namespace.

Returns

`true` if this element has an explicitly specified value or a default value for the specified attribute; `false` otherwise.

Description

This method works like `hasAttribute( )`, except that the attribute to be checked for is specified by namespace and name. This method is useful only with XML documents that use namespaces.

See Also

Element.getAttributeNS( ), Element.setAttributeNS( )

Element.removeAttribute( )

DOM Level 1 Core

delete a named attribute of an element

Synopsis

```
void removeAttribute(String name);
```

Arguments

name
: The name of the attribute to be deleted.

Throws

This method may throw a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if this element is read-only and does not allow its attributes to be removed.

Description

`removeAttribute( )` deletes a named attribute from this element. If the named attribute has a default value specified by the document type, subsequent calls to `getAttribute( )` will return that default value. Attempts to remove nonexistent attributes or attributes that are not specified but have a default value are silently ignored.

See Also

Element.getAttribute( ), Element.setAttribute( ), Node.attributes

Element.removeAttributeNode( )

DOM Level 1 Core

remove an Attr node from an element

Synopsis

```
Attr removeAttributeNode(Attr oldAttr)
    throws DOMException;
```

Arguments

oldAttr
: The Attr node to be removed from the element.

Returns

The Attr node that was removed.

Throws

This method may throw a DOMException with the following code values:

`NO_MODIFICATION_ALLOWED_ERR`
: This element is read-only and does not allow attributes to be removed.

`NOT_FOUND_ERR`
: *oldAttr* is not an attribute of this element.

Description

This method removes (and returns) an Attr node from the set of attributes of an element. If the removed attribute has a default value specified by the DTD, a new Attr is added representing this default value. If is often simpler to use `removeAttribute( )` instead of this method.

See Also

Attr, Element.removeAttribute( )

Element.removeAttributeNS( )

DOM Level 2 Core

delete an attribute specified by name and namespace

Synopsis

```
void removeAttributeNS(String namespaceURI,
                       String localName);
```

Arguments

namespaceURI
: The unique identifier of the namespace of the attribute, or `null` for no namespace.

localName
: The name of the attribute within the specified namespace.

Throws

This method may throw a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if this element is read-only and does not allow its attributes to be removed.

Description

`removeAttributeNS( )` works just like `removeAttribute( )`, except that the attribute to be removed is specified by name and namespace instead of simply by name. This method is useful only with XML documents that use namespaces.

See Also

Element.getAttributeNS( ), Element.removeAttribute( ), Element.setAttributeNS( )

Element.setAttribute( )

DOM Level 1 Core

create or change an attribute of an element

Synopsis

```
void setAttribute(String name,
                  String value)
    throws DOMException;
```

Arguments

name
: The name of the attribute that is to be created or modified.

value
: The string value of the attribute.

Throws

This method may throw a DOMException with the following `code` values:

`INVALID_CHARACTER_ERR`
: The *name* argument contains a character that is not allowed in HTML or XML attribute names.

`NO_MODIFICATION_ALLOWED_ERR`
: This element is read-only and does not allow modifications to its attributes.

Description

This method sets the specified attribute to the specified value. If no attribute by that name already exists, a new one is created. Note that Element objects that represent the tags of an HTML document also implement the HTMLElement interface and (usually) one of its tag-specific subinterfaces. As a shortcut, these interfaces define properties that correspond to the standard HTML attributes for each tag, and it is usually easier to set an HTML attribute simply by setting the appropriate property.

The *value* argument is a plain string. If you are working with an XML document and need to include an entity reference in an attribute value, use `setAttributeNode( )`.

Example

```
// Set the TARGET attribute of all links in a document
var links = document.body.getElementsByTagName("A");
for(var i = 0; i < links.length; i++) {
    links[i].setAttribute("TARGET", "newwindow");
}
```

See Also

Element.getAttribute( ), Element.removeAttribute( ), Element.setAttributeNode( )

Element.setAttributeNode( )

DOM Level 1 Core

add a new Attr node to an Element

Synopsis

```
Attr setAttributeNode(Attr newAttr)
    throws DOMException;
```

Arguments

newAttr
: The Attr node that represents the attribute to be added or whose value is to be modified.

Returns

The Attr node that was replaced by *newAttr*, or `null` if no attribute was replaced.

Throws

This method may throw a DOMException with a `code` of the following values:

`INUSE_ATTRIBUTE_ERR`
: *newAttr* is already a member of the attribute set of some other Element node.

`NO_MODIFICATION_ALLOWED_ERR`
: The Element node is read-only and does not allow modifications to its attributes.

`WRONG_DOCUMENT_ERR`
: *newAttr* has a different `ownerDocument` property than the Element on which it is being set.

Description

This method adds a new Attr node to the set of attributes of an Element node. If an attribute with the same name already exists for the Element, *newAttr* replaces that attribute, and the replaced Attr node is returned. If no such attribute already exists, this method defines a new attribute for the Element.

It is usually easier to use `setAttribute( )` instead of `setAttributeNode( )`. However, you should use `setAttributeNode( )` when you need to define an attribute whose value contains an entity reference for an XML document.

See Also

Attr, Element.setAttribute( )

Element.setAttributeNodeNS( )

DOM Level 2 Core

add a namespace Attr node to an Element

Synopsis

```
Attr setAttributeNodeNS(Attr newAttr)
    throws DOMException;
```

Arguments

newAttr
: The Attr node that represents the attribute to be added or whose value is to be modified.

Returns

The Attr node that was replaced by *newAttr*, or `null` if no attribute was replaced.

Throws

This method throws exceptions for the same reasons as `setAttributeNode( )`. It may also throw a DOMException with a `code` of `NOT_SUPPORTED_ERR` to signal that the method is not implemented because the current implementation does not support XML documents and namespaces.

Description

This method works just like `setAttributeNode( )`, except that it is designed for use with Attr nodes that represent attributes specified by namespace and name.

This method is useful only with XML documents that use namespaces. It may be unimplemented (i.e., throw a `NOT_SUPPORTED_ERR`) on browsers that do not support XML documents.

See Also

Attr, Element.setAttributeNS( ), Element.setAttributeNode( )

Element.setAttributeNS( )

DOM Level 2 Core

create or change an attribute with a namespace

Synopsis

```
void setAttributeNS(String namespaceURI,
                    String qualifiedName,
                    String value)
    throws DOMException;
```

Arguments

namespaceURI
: The URI that uniquely identifies the namespace of the attribute to be set or created, or `null` for no namespace.

qualifiedName
: The name of the attribute, specified as a namespace prefix followed by a colon and a name within the namespace.

value
: The new value of the attribute.

Throws

This method may throw a DOMException with the following code values:

INVALID_CHARACTER_ERR
: The *qualifiedName* argument contains a character that is not allowed in HTML or XML attribute names.

NAMESPACE_ERR
: *qualifiedName* is malformed, or there is a mismatch between the namespace prefix of *qualifiedName* and the *namespaceURI* argument.

NO_MODIFICATION_ALLOWED_ERR
: This element is read-only and does not allow modifications to its attributes.

NOT_SUPPORTED_ERR
: The DOM implementation does not support XML documents.

Description

This method is like setAttribute( ), except that the attribute to be created or set is specified by a namespace URI and a qualified name that consists of a namespace prefix, a colon, and a local name within the namespace. In addition to letting you change the value of an attribute, this method allows you to change the namespace prefix of an attribute.

This method is useful only with XML documents that use namespaces. It may be unimplemented (i.e., throw a NOT_SUPPORTED_ERR) on browsers that do not support XML documents.

See Also

Element.setAttribute( ), Element.setAttributeNode( )

ElementCSSInlineStyle

see HTMLElement

Entity

DOM Level 1 XML

an entity in an XML DTD

Node → Entity

Properties

readonly String notationName
: The notation name (for unparsed entities), or null if there is none (for parsed entities). See the notations property of DocumentType for a way to look up a Notation node by name.

readonly String publicId
: The public identifier for this entity, or null if none was specified.

`readonly String systemId`
The system identifier for this entity, or `null` if none was specified.

Description

This infrequently used interface represents an entity in an XML document type definition (DTD). It is never used with HTML documents.

The name of the entity is specified by the `nodeName` property inherited from the Node interface. The entity content is represented by the child nodes of the Entity node. Note that Entity nodes and their children are not part of the document tree (and the `parentNode` property of an Entity is always `null`). Instead, a document may contain one or more references to an entity; see "EntityReference" for more information.

Entities are defined in the DTD of a document, either as part of an external DTD file or as part of an "internal subset" that defines local entities specific to the current document. The DocumentType interface has an `entities` property that allows Entity nodes to be looked up by name. This is the only way to obtain a reference to an Entity node; because they are part of the document type, Entity nodes never appear as part of the document itself.

Entity nodes and all of their descendants are read-only and cannot be edited or modified in any way.

See Also

DocumentType, EntityReference, Notation

EntityReference

DOM Level 1 XML

a reference to an entity defined in an XML DTD

Node → EntityReference

Description

This infrequently used interface represents a reference from an XML document to an entity defined in the document's DTD. Character entities and predefined entities such as `<` are always expanded in XML and HTML documents, and EntityReference nodes never appear in HTML documents, so programmers working exclusively with HTML documents never need to use this interface. Note also that some XML parsers expand all entity references. Documents created by such parsers do not contain EntityReference nodes.

This interface defines no properties or methods of its own. The inherited `nodeName` property specifies the name of the referenced entity. The `entities` property of the DocumentType interface provides a way to look up the Entity object with that name. Note, however, that the DocumentType may not contain an Entity node with the specified name (because, for example, nonvalidating XML parsers are not required to parse the "external subset" of the DTD). In this case, the EntityReference has no children. On the other hand, if the DocumentType does contain an Entity node with the specified name, the child nodes of the EntityReference are copies of the child nodes of the Entity node and represent the content of the entity. Like Entity nodes, EntityReference nodes and their descendants are read-only and cannot be edited or modified.

See Also

DocumentType

Returned by: Document.createEntityReference( )

Event

DOM Level 2 Events

information about an event

Subinterfaces

MutationEvent, UIEvent

Constants

These constants are the legal values of the eventPhase property; they represent the current phase of event propagation for this event:

`unsigned short CAPTURING_PHASE = 1`
: The event is in its capturing phase.

`unsigned short AT_TARGET = 2`
: The event is being handled by its target node.

`unsigned short BUBBLING_PHASE = 3`
: The event is bubbling.

Properties

`readonly boolean bubbles`
: `true` if the event is of a type that bubbles (unless `stopPropagation( )` is called); `false` otherwise.

`readonly boolean cancelable`
: `true` if the default action associated with the event can be canceled with `preventDefault( )`; `false` otherwise.

`readonly EventTarget currentTarget`
: The Document node that is currently handling this event. During capturing and bubbling, this is different from `target`. Note that all nodes implement the EventTarget interface, and the `currentTarget` property may refer to any node; it is not restricted to Element nodes.

`readonly unsigned short eventPhase`
: The current phase of event propagation. The three previous constants define the legal values for this property.

`readonly EventTarget target`
: The target node for this event; i.e., the node that generated the event. Note that this may be any node; it is not restricted to Element nodes.

`readonly Date timeStamp`
: The date and time at which the event occurred (or, technically, at which the Event object was created). Implementations are not required to provide valid time data in

this field, and if they do not, the `getTime( )` method of this Date object should return 0. See the Date object in the core reference section of this book.

`readonly String type`
: The name of the event that this Event object represents. This is the name under which the event handler was registered, or the name of the event handler property with the leading "on" removed. For example, "click", "load", or "submit". See Table 19-3 in Chapter 19 for a complete list of event types defined by the DOM standard.

Methods

`initEvent( )`
: Initializes the properties of a newly created Event object.

`preventDefault( )`
: Tells the web browser not to perform the default action associated with this event, if there is one. If the event is not of a type that is cancelable, this method has no effect.

`stopPropagation( )`
: Stops the event from propagating any further through the capturing, target, or bubbling phases of event propagation. After this method is called, any other event handlers for the same event on the same node will be called, but the event will not be dispatched to any other nodes.

Description

This interface represents an event that occurred on some node of the document and contains details about the event. Various subinterfaces of Event define additional properties that provide details pertinent to specific types of events.

Many event types use a more specific subinterface of Event to describe the event that has occurred. However, the event types defined by the HTMLEvents module use the Event interface directly. These event types are: "abort", "blur", "change", "error", "focus", "load", "reset", "resize", "scroll", "select", "submit", and "unload".

See Also

EventListener, EventTarget, MouseEvent, UIEvent; Chapter 19

Passed to: EventTarget.dispatchEvent( )

Returned by: Document.createEvent( )

Event.initEvent( )

DOM Level 2 Events

initialize the properties of a new Event

Synopsis

```
void initEvent(String eventTypeArg,
               boolean canBubbleArg,
               boolean cancelableArg);
```

Arguments

eventTypeArg
: The type of event. This may be one of the predefined event types, such as "load" or "submit", or it may be a custom type of your own choosing. Names that begin with "DOM" are reserved, however.

canBubbleArg
: Whether the event will bubble.

cancelableArg
: Whether the event may be canceled with `preventDefault( )`.

Description

This method initializes the `type`, `bubbles`, and `cancelable` properties of a synthetic Event object created by `Document.createEvent( )`. This method may be called on newly created Event objects only before they have been dispatched with the `EventTarget.dispatchEvent( )` method.

Event.preventDefault( )

DOM Level 2 Events

cancel default action of an event

Synopsis

```
void preventDefault( );
```

Description

This method tells the web browser not to perform the default action (if any) associated with this event. For example, if the `type` property is "submit", any event handler called during any phase of event propagation can prevent the form submission by calling this method. Note that if the `cancelable` property of an Event object is `false`, either there is no default action or there is a default action that cannot be prevented. In either case, calling this method has no effect.

Event.stopPropagation( )

DOM Level 2 Events

do not dispatch an event any further

Synopsis

```
void stopPropagation( );
```

Description

This method stops the propagation of an event and prevents it from being dispatched to any other Document nodes. It may be called during any phase of event propagation. Note that this method does not prevent other event handlers on the same Document node from being called, but it does prevent the event from being dispatched to any other nodes.

EventException

DOM Level 2 Events

signals an event-specific exception or error

Constants

The following constant defines the one legal value for the `code` property of an EventException object. Note that this constant is a static property of EventException, not a property of individual exception objects.

`unsigned short UNSPECIFIED_EVENT_TYPE_ERR = 0`

An Event object has a `type` property that is uninitialized, or is `null` or the empty string.

Properties

`unsigned short code`

An error code that provides some detail about what caused the exception. In the Level 2 DOM there is only one possible value for this field, defined by the constant above.

Description

An EventException is thrown by certain event-related methods to signal a problem of some sort. (In the DOM Level 2 specification, an exception of this type is thrown only by `EventTarget.dispatchEvent()`).

EventListener

DOM Level 2 Events

an event handler function

Methods

`handleEvent()`

In languages such as Java that do not allow functions to be passed as arguments to other functions, you define an event listener by defining a class that implements this interface and includes an implementation for this `handleEvent()` method. When an event occurs, the system calls this method and passes in an Event object that describes the event.

In JavaScript, however, you define an event handler simply by writing a function that accepts an Event object as its argument. The name of the function does not matter, and the function itself is used in place of an EventListener object. See the "Example" section.

Description

This interface defines the structure of an event listener or event handler. In languages such as Java, an event listener is an object that defines a method named `handleEvent()` that takes an Event object as its sole argument. In JavaScript, however, any function that expects a single Event argument, or a function that expects no argument, can serve as an event listener.

Example

```
// This function is an event listener for a "submit" event
function submitHandler(e) {
```

```
  // Call a form-validation function defined elsewhere
  if (!validate(e.target))
      e.preventDefault( );  // If validation fails, don't submit form
}

// We might register the event listener above like this
document.forms[0].addEventListener("submit", submitHandler, false);
```

See Also

Event, EventTarget; Chapter 19

Passed to: EventTarget.addEventListener( ), EventTarget.removeEventListener( )

EventTarget

DOM Level 2 Events

event listener registration methods

Methods

`addEventListener( )`
: Adds an event listener to the set of event listeners for this node.

`dispatchEvent( )`
: Dispatches a synthetic event to this node.

`removeEventListener( )`
: Removes an event listener from the set of listeners of this Document node.

Description

In DOM implementations that support events (i.e., those that support the "Events" feature), all nodes in the document tree implement this interface and maintain a set or list of event listener functions for each node. The `addEventListener( )` and `removeEventListener( )` methods allow listener functions to be added and removed from this set.

See Also

Event, EventListener; Chapter 19

Type of: Event.currentTarget, Event.target, MouseEvent.relatedTarget

Passed to: MouseEvent.initMouseEvent( )

EventTarget.addEventListener( )

DOM Level 2 Events

register an event handler

Synopsis

```
void addEventListener(String type,
                      EventListener listener,
                      boolean useCapture);
```

Arguments

type
: The type of event for which the event listener is to be invoked. For example, "load", "click", or "mousedown".

listener
: The event listener function that will be invoked when an event of the specified type is dispatched to this Document node.

useCapture
: If `true`, the specified *`listener`* is to be invoked only during the capturing phase of event propagation. The more common value of `false` means that the *`listener`* will not be invoked during the capturing phase but instead will be invoked when this node is the actual event target or when the event bubbles up to this node from its original target.

Description

This method adds the specified event listener function to the set of listeners registered on this node to handle events of the specified *`type`*. If *`useCapture`* is `true`, the listener is registered as a capturing event listener. If *`useCapture`* is `false`, it is registered as a normal event listener.

`addEventListener( )` may be called multiple times to register multiple event handlers for the same type of event on the same node. Note, however, that the DOM makes no guarantees about the order in which multiple event handlers will be invoked.

If the same event listener function is registered twice on the same node with the same *`type`* and *`useCapture`* arguments, the second registration is simply ignored. If a new event listener is registered on this node while an event is being handled at this node, the new event listener is not invoked for that event.

When a Document node is duplicated with `Node.cloneNode( )` or `Document.importNode( )`, the event listeners registered for the original node are not copied.

See Also

Event, EventListener; Chapter 19

EventTarget.dispatchEvent( )

DOM Level 2 Events

dispatch a synthetic event to this node

Synopsis

```
boolean dispatchEvent(Event evt)
    throws EventException;
```

Arguments

evt
: The Event object to be dispatched.

Returns

`false` if the `preventDefault( )` method of *evt* was called at any time during the propagation of the event, or `true` otherwise.

Throws

This method throws an `EventException` with its `code` property set to `EventException.UNSPECIFIED_EVENT_TYPE_ERR` if the Event object *evt* was not initialized or if its `type` property was `null` or the empty string.

Description

This method dispatches a synthetic event created with `Document.createEvent( )` and initialized with the initialization method defined by the Event interface or one of its subinterfaces. The node on which this method is called becomes the target of the event, but the event first propagates down the document tree during the capturing phase, and then, if the `bubbles` property of the event is `true`, it bubbles up the document tree after being handled at the event target itself.

See Also

Document.createEvent( ), Event.initEvent( ), MouseEvent.initMouseEvent( )

EventTarget.removeEventListener( )

DOM Level 2 Events

delete an event listener

Synopsis

```
void removeEventListener(String type,
                         EventListener listener,
                         boolean useCapture);
```

Arguments

type
: The type of event for which the event listener is to be deleted.

listener
: The event listener function that is to be removed.

useCapture
: `true` if a capturing event listener is to be removed; `false` if a normal event listener is to be removed.

Description

This method removes the specified event listener function. The *type* and *useCapture* arguments must be the same as they were in the corresponding call to `addEventListener( )`. If no event listener is found that matches the specified arguments, this method does nothing.

Once an event listener function has been removed by this method, it will no longer be invoked for the specified *type* of event on this node. This is true even if the event listener is removed by another event listener registered for the same type of event on the same node.

HTMLAnchorElement

DOM Level 1 HTML

a hyperlink or anchor in an HTML document

Node → Element → HTMLElement → HTMLAnchorElement

Properties

This interface defines the properties in the following table, which correspond to the HTML attributes of the <a> tag.

Property	Attribute	Description
String accessKey	accesskey	Keyboard shortcut
String charset	charset	Encoding of the destination document
String coords	coords	Used inside <map> elements
String href	href	URL of the hyperlink
String hreflang	hreflang	Language of the linked document
String name	name	Name of the anchor
String rel	rel	Link type
String rev	rev	Reverse link type
String shape	shape	Used inside <map> elements
long tabIndex	tabindex	Link's position in tabbing order
String target	target	Name of the frame or window in which the destination document is to be displayed
String type	type	Content type of the destination document

Methods

`blur( )`
Takes keyboard focus away from the link.

`focus( )`
Scrolls the document so the anchor or link is visible and gives keyboard focus to the link.

Description

This interface represents an <a> tag in an HTML document. `href`, `name`, and `target` are the key properties, representing the most commonly used attributes of the tag.

HTMLAnchorElement objects can be obtained from the `links` and `anchors` HTMLCollection properties of the HTMLDocument interface.

Example

```
// Get the destination URL of first the hyperlink in the document
var url = document.links[0].href;
// Scroll the document so the anchor named "_bottom_" is visible
document.anchors['_bottom_'].focus();
```

See Also

Link and Anchor objects in the client-side reference section

HTMLAnchorElement.blur( )

DOM Level 1 HTML

take keyboard focus away from a hyperlink

Synopsis

```
void blur( );
```

Description

For web browsers that allow hyperlinks to have the keyboard focus, this method takes keyboard focus away from a hyperlink.

HTMLAnchorElement.focus( )

DOM Level 1 HTML

make a link or anchor visible and give it keyboard focus

Synopsis

```
void focus( );
```

Description

This method scrolls the document so the specified anchor or hyperlink is visible. If the element is a hyperlink and the browser allows hyperlinks to have keyboard focus, this method also gives keyboard focus to the element.

HTMLBodyElement

DOM Level 1 HTML

the <body> of an HTML document

Node → Element → HTMLElement → HTMLBodyElement

Properties

`deprecated String aLink`
: The value of the `alink` attribute. Specifies the color of "active" links; that is, the color of a link when the mouse has been pressed over it but has not yet been released.

`deprecated String background`
: The value of the `background` attribute. Specifies the URL of an image to use as a background texture for the document.

`deprecated String bgColor`
: The value of the `bgcolor` attribute. Specifies the background color of the document.

`deprecated String link`
: The value of the `link` attribute. Specifies the normal color of unvisited hyperlinks.

deprecated String text
: The value of the text attribute. Specifies the foreground color (the color of text) for the document.

deprecated String vLink
: The value of the vlink attribute. Specifies the normal color of "visited" hyperlinks that have already been followed.

Description

The HTMLBodyElement interface represents the <body> tag of a document. All HTML documents have a <body> tag, even if it does not explicitly appear in the document source. You can obtain the HTMLBodyElement of a document through the body property of the HTMLDocument interface.

The properties of this object specify default colors and images for the document. Although these properties represent the values of <body> attributes, the Level 0 DOM made these same values accessible through properties (with different names) of the Document object. See the Document object in the client-side reference section of this book for details.

Although these color and image properties belong more appropriately to the HTMLBodyElement interface than they do to the Document object, note that they are all deprecated because the HTML 4 standard deprecates the <body> attributes that they represent. The preferred way to specify colors and images for a document is using CSS styles.

Example

```
document.body.text = "#ff0000";         // Display text in bright red
document.fgColor = "#ff0000";           // Same thing using old DOM Level 0 API
document.body.style.color = "#ff0000";  // Same thing using CSS styles
```

See Also

Document object in the client-side reference section

HTMLCollection

DOM Level 1 HTML

array of HTML elements accessible by position or name

Properties

readonly unsigned long length
: The number of elements in the collection.

Methods

item( )
: Returns the element at the specified position in the collection. You can also simply specify the position within array brackets instead of calling this method explicitly.

namedItem( )
: Returns the element from the collection that has the specified value for its id or name attribute, or null if there is no such element. You may also place the element name within array brackets instead of calling this method explicitly.

Description

An HTMLCollection is a collection of HTML elements with methods that allow you to retrieve the elements by their position in the document or by their `id` or `name` attribute. In JavaScript, HTMLCollection objects behave like read-only arrays, and you may use JavaScript square-bracket notation to index an HTMLCollection by number or by name instead of calling the `item( )` and `namedItem( )` methods.

A number of the properties of the HTMLDocument interface (which standardizes the DOM Level 0 Document object) are HTMLCollection objects, which provide convenient access to document elements such as forms, images, and links. The HTMLCollection object also provides a convenient way to traverse the elements of an HTML form, the rows of an HTML table, the cells of a table row, and the areas of a client-side image map.

HTMLCollection objects are read-only: you cannot assign new elements to them, even when using JavaScript array notation. They are "live," meaning that if the underlying document changes, those changes are immediately visible through all HTMLCollection objects.

Example

```
var c = document.forms;        // This is an HTMLCollection of form elements
var firstform = c[0];          // It can be used like a numeric array
var lastform = c[c.length-1];  // The length property gives the number of elements
var address = c["address"];    // It can be used like an associative array
var address = c.address;       // JavaScript allows this notation, too
```

See Also

NodeList

Type of: HTMLDocument.anchors, HTMLDocument.applets, HTMLDocument.forms, HTMLDocument.images, HTMLDocument.links, HTMLFormElement.elements, HTMLMapElement.areas, HTMLSelectElement.options, HTMLTableElement.rows, HTMLTableElement.tBodies, HTMLTableRowElement.cells, HTMLTableSectionElement.rows

HTMLCollection.item( )

DOM Level 1 HTML

get an element by position

Synopsis

```
Node item(unsigned long index);
```

Arguments

index

The position of the element to be returned. Elements appear in an HTMLCollection in the same order in which they appear in the document source.

Returns

The element at the specified *index*, or `null` if *index* is less than zero or greater than or equal to the `length` property.

Description

The item( ) method returns a numbered element from an HTMLCollection. In JavaScript, it is easier to treat the HTMLCollection as an array and to index it using array notation.

Example

```
var c = document.images;  // This is an HTMLCollection
var img0 = c.item(0);     // You can use the item( ) method this way
var img1 = c[1];          // But this notation is easier and more common
```

See Also

NodeList.item( )

HTMLCollection.namedItem( )

DOM Level 1 HTML

get an element by name

Synopsis

```
Node namedItem(String name);
```

Arguments

name
The name of the element to be returned.

Returns

An element with the specified *name*, or null if no elements in the HTMLCollection have that name.

Description

This method finds and returns an element from the HTMLCollection that has the specified name. If any element has an id attribute whose value is the specified name, that element is returned. If no such element is found, an element whose name attribute has the specified value is returned. If no such element exists, namedItem( ) returns null.

Note that any HTML element may be given an id attribute, but only certain HTML elements—such as forms, form elements, images, and anchors—may have a name attribute.

In JavaScript, it is easier to treat the HTMLCollection as an associative array and to specify *name* between square brackets using array notation.

Example

```
var forms = document.forms;                // An HTMLCollection of forms
var address = forms.namedItem("address");  // Finds <form name="address">
var payment = forms["payment"]  // Simpler syntax: finds <form name="payment">
var login = forms.login;        // Also works: finds <form name="login">
```

HTMLDocument

DOM Level 1 HTML

the root of an HTML document tree

Node → Document → HTMLDocument

Properties

`readonly HTMLCollection anchors`
: An array (HTMLCollection) of all anchors in the document. For compatibility with The Level 0 DOM, this array contains only `<a>` elements that have a `name` attribute specified; it does not include anchors created with an `id` attribute.

`readonly HTMLCollection applets`
: An array (HTMLCollection) of all applets in a document. These include applets defined with an `<object>` tag and all `<applet>` tags.

`HTMLElement body`
: A convenience property that refers to the HTMLBodyElement that represents the `<body>` tag of this document. For documents that define framesets, this property refers to the outermost `<frameset>` tag.

`String cookie`
: Allows cookies to be queried and set for this document. See "Document.cookie" in the client-side reference section.

`readonly String domain`
: The domain name of the server from which the document was loaded, or `null` if there is none. Contrast with the read/write `Document.domain` property in the client-side reference section.

`readonly HTMLCollection forms`
: An array (HTMLCollection) of all HTMLFormElement objects in the document.

`readonly HTMLCollection images`
: An array (HTMLCollection) of all `<img>` tags in the document. Note that for compatibility with the Level 0 DOM, images defined with an `<object>` tag are not included in this collection.

`readonly HTMLCollection links`
: An array (HTMLCollection) of all hyperlinks in the document. These include all `<a>` tags that have an `href` attribute, and all `<area>` tags.

`readonly String referrer`
: The URL of the document that linked to this document, or `null` if this document was not accessed through a hyperlink.

`String title`
: The contents of the `<title>` tag for this document.

`readonly String URL`
: The URL of the document.

Methods

`close( )`
: Closes a document stream opened with the `open( )` method, forcing any buffered output to be displayed.

`getElementById( )`
: Returns the element with the specified `id`. In the Level 2 DOM, this method is inherited from the Document interface.

`getElementsByName( )`
: Returns an array of nodes (a NodeList) of all elements in the document that have a specified value for their `name` attribute.

`open( )`
: Opens a stream to which new document contents may be written. Note that this method erases any current document content.

`write( )`
: Appends a string of HTML text to an open document.

`writeln( )`
: Appends a string of HTML text followed by a newline character to an open document.

Description

This interface extends Document and defines HTML-specific properties and methods that provide compatibility with the DOM Level 0 Document object (see the Document object in the client-side reference section). Note that HTMLDocument does not have all the properties of the Level 0 Document object. The properties that specify document colors and background images have been renamed and moved to the HTMLBodyElement.

Finally, note that in the Level 1 DOM, HTMLDocument defines a method named `getElementById( )`. In the Level 2 DOM, this method has been moved to the Document interface, and it is now inherited by HTMLDocument rather than defined by it. See the "Document.getElementById( )" entry in this reference section for details.

See Also

Document.getElementById( ), HTMLBodyElement; Document object in the client-side reference section

Returned by: HTMLDOMImplementation.createHTMLDocument( )

HTMLDocument.close( )

DOM Level 1 HTML

close an open document and display it

Synopsis

```
void close( );
```

Description

This method closes a document stream that was opened with the `open( )` method and forces any buffered output to be displayed. See the "Document.close( )" entry in the client-side reference section for full details.

See Also

HTMLDocument.open( ); Document.close( ) in the client-side reference section

HTMLDocument.getElementById( )

see Document.getElementById( )

HTMLDocument.getElementsByName( ) DOM Level 1 HTML

find elements with the specified name attribute

Synopsis

```
Node[] getElementsByName(String elementName);
```

Arguments

elementName
The desired value for the name attribute.

Returns

An array (really a NodeList) of elements that have a name attribute of the specified value. If no such elements are found, the returned NodeList is empty and has a length of 0.

Description

This method searches an HTML document tree for Element nodes that have a name attribute of the specified value and returns a NodeList (which you can treat as an array) containing all matching elements. If there are no matching elements, a NodeList with length 0 is returned.

Do not confuse this method with the Document.getElementById( ) method, which finds a single Element based on the unique value of an id attribute, or with the Document.getElementsByTagName( ) method, which returns a NodeList of elements with the specified tag name.

See Also

Document.getElementById( ), Document.getElementsByTagName( )

HTMLDocument.open( ) DOM Level 1 HTML

begin a new document, erasing the current one

Synopsis

```
void open( );
```

Description

This method erases the current HTML document and begins a new one, which may be written to with the write( ) and writeln( ) methods. After calling open( ) to begin a new document and write( ) to specify document content, you must always remember to call close( ) to end the document and force its content to be displayed.

This method should not be called by a script or event handler that is part of the document being overwritten, since the script or handler will itself be overwritten.

See "Document.open()" in the client-side reference section, but note that this standardized version of that method can be used only to create new HTML documents and does not accept the optional *mimetype* argument that the Level 0 version does.

Example

```
var w = window.open("");          // Open a new window
var d = w.document;               // Get its HTMLDocument object
d.open();                         // Open the document for writing
d.write("<h1>Hello world</h1>");  // Output some HTML to the document
d.close();                        // End the document and display it
```

See Also

HTMLDocument.close(), HTMLDocument.write(); Document.open() in the client-side reference section

HTMLDocument.write( )

DOM Level 1 HTML

append HTML text to an open document

Synopsis

```
void write(String text);
```

Arguments

text
: The HTML text to be appended to the document.

Description

This method appends the specified HTML text to the document, which must have been opened with the `open( )` method and must not yet have been closed with `close( )`.

See "Document.write()" in the client-side reference section for complete details, but note that this standardized version of that Level 0 method accepts only a single string argument, not an arbitrary number of arguments.

See Also

HTMLDocument.open(); Document.write() in the client-side reference section

HTMLDocument.writeln( )

DOM Level 1 HTML

append HTML text and a newline to an open document

Synopsis

```
void writeln(String text);
```

Arguments

text
: The HTML text to be appended to the document.

Description

This method is like `HTMLDocument.write( )`, except that it follows the appended text with a newline character, which can be useful when writing the content of a `<pre>` tag, for example.

See Also

Document.writeln( ) in the client-side reference section

HTMLDOMImplementation

see DOMImplementation

HTMLElement

DOM Level 1 HTML

the base interface for all HTML elements

Node → Element → HTMLElement

Also Implements

ElementCSSInlineStyle

If the implementation supports CSS style sheets, all objects that implement this interface also implement the ElementCSSInlineStyle interface. Because CSS support is quite common in web browsers, the `style` property defined by that interface is included here for convenience.

Subinterfaces

HTMLAnchorElement
HTMLAppletElement
HTMLAreaElement
HTMLBRElement
HTMLBaseElement
HTMLBaseFontElement
HTMLBodyElement
HTMLButtonElement
HTMLDListElement
HTMLDirectoryElement
HTMLDivElement
HTMLFieldSetElement
HTMLFontElement
HTMLFormElement
HTMLFrameElement
HTMLFrameSetElement
HTMLHRElement
HTMLHeadElement
HTMLHeadingElement
HTMLHtmlElement
HTMLIFrameElement
HTMLImageElement
HTMLInputElement
HTMLIsIndexElement
HTMLLIElement
HTMLLabelElement
HTMLLegendElement
HTMLLinkElement
HTMLMapElement
HTMLMenuElement
HTMLMetaElement
HTMLModElement
HTMLOListElement
HTMLObjectElement
HTMLOptGroupElement
HTMLOptionElement
HTMLParagraphElement
HTMLParamElement
HTMLPreElement
HTMLQuoteElement
HTMLScriptElement
HTMLSelectElement
HTMLStyleElement
HTMLTableCaptionElement
HTMLTableCellElement
HTMLTableColElement
HTMLTableElement
HTMLTableRowElement
HTMLTableSectionElement
HTMLTextAreaElement
HTMLTitleElement
HTMLUListElement

Properties

`readonly CSS2Properties style`

The value of the `style` attribute that specifies inline CSS styles for this element. This property is not actually defined directly by the HTMLElement interface; instead, it is

defined by the ElementCSSInlineStyle interface. If a browser supports CSS style sheets, all of its HTMLElement objects implement ElementCSSInlineStyle and define this `style` property. The value of this property is an object that implements the CSSStyleDeclaration interface and the (more commonly used) CSS2Properties interface.

`String className`
: The value of the `class` attribute of the element, which specifies the name of a CSS class. Note that this property is not named "class" because that name is a reserved word in JavaScript.

`String dir`
: The value of the `dir` attribute of the element, which specifies the text direction for the document.

`String id`
: The value of the `id` attribute. No two elements within the same document should have the same value for `id`.

`String lang`
: The value of the `lang` attribute, which specifies the language code for the document.

`String title`
: The value of the `title` attribute, which specifies descriptive text suitable for display in a "tooltip" for the element.

Description

This interface defines properties that represent the attributes shared by all HTML elements. All HTML elements implement this interface, and most implement a tag-specific subinterface that defines properties for each of that tag's attributes. In addition to the properties listed here, see the "HTMLElement" reference page in the client-side reference section for a list of DOM Level 0 event handler properties that are supported by all HTML elements in a document.

Some HTML tags do not allow any attributes other than the universal attributes allowed on all HTML tags and represented by the properties of HTMLElement. These tags do not have their own tag-specific subinterface, and elements of this type in the document tree are represented by an HTMLElement object. The tags without a tag-specific interface of their own are the following:

`<abbr>`	`<acronym>`	`<address>`	`<b>`
`<bdo>`	`<big>`	`<center>`	`<cite>`
`<code>`	`<dd>`	`<dfn>`	`<dt>`
`<em>`	`<i>`	`<kbd>`	`<noframes>`
`<noscript>`	`<s>`	`<samp>`	`<small>`
`<span>`	`<strike>`	`<strong>`	`<sub>`
`<sup>`	`<tt>`	`<u>`	`<var>`

As you can see from the earlier "Subinterfaces" section, there are many HTML tags that *do* have tag-specific subinterfaces. Typically, a tag named *T* has a tag-specific interface named HTML*T*Element. For example, the `<head>` tag is represented by the HTMLHeadElement interface. In a few cases, two or more related tags share a single interface, as in

the case of the <h1> through <h6> tags, which are all represented by the HTMLHeadingElement interface.

Most of these tag-specific interfaces do nothing more than define a JavaScript property for each attribute of the HTML tag. The JavaScript properties have the same names as the attributes and use lowercase (e.g., id) or, when the attribute name consists of multiple words, mixed-case (e.g., className). When an HTML attribute name is a reserved word in Java or JavaScript, the property name is changed slightly. For example, the class attribute of all HTML tags becomes the className property of the HTMLElement interface because class is a reserved word. Similarly, the for attribute of <label> and <script> tags becomes the htmlFor property of the HTMLLabelElement and HTMLScriptElement interfaces because for is a reserved word. The meanings of those properties that correspond directly to HTML attributes are defined by the HTML specification, and documenting each one is beyond the scope of this book.

The following table lists all the HTML tags that have a corresponding subinterface of HTMLElement. For each tag, the table lists the interface name and the names of the properties and methods it defines. All properties are read/write strings unless otherwise specified. For properties that are not read/write strings, the property type is specified in square brackets before the property name. Because these interfaces and their properties map so directly to HTML elements and attributes, most interfaces do not have reference pages of their own in this book, and you should consult an HTML reference for details. The exceptions are interfaces that define methods and interfaces that represent certain particularly important tags, such as the <body> tag. These interfaces are marked with a * in the table, and you can look them up in this reference section for further details.

HTML tag	DOM interface, properties, and methods
all tags	HTMLElement*: id, title, lang, dir, className
<a>	HTMLAnchorElement*: accessKey, charset, coords, href, hreflang, name, rel, rev, shape, [long] tabIndex, target, type, blur( ), focus( )
<applet>	HTMLAppletElement†: align†, alt†, archive†, code†, codeBase†, height†, hspace†, name†, object†, vspace†, width†
<area>	HTMLAreaElement: accessKey, alt, coords, href, [boolean] noHref, shape, [long] tabIndex, target
<base>	HTMLBaseElement: href, target
<basefont>	HTMLBaseFontElement†: color†, face†, size†
<blockquote>, <q>	HTMLQuoteElement: cite
<body>	HTMLBodyElement*: aLink†, background†, bgColor†, link†, text†, vLink†
 	HTMLBRElement: clear†
<button>	HTMLButtonElement: [readonly HTMLFormElement] form, accessKey, [boolean] disabled, name, [long] tabIndex, [readonly] type, value
<caption>	HTMLTableCaptionElement*: align†
<col>, <colgroup>	HTMLTableColElement*: align, ch, chOff, [long] span, vAlign, width
<del>, <ins>	HTMLModElement: cite, dateTime

HTML tag	DOM interface, properties, and methods
<dir>	HTMLDirectoryElement†: [boolean] compact†
	HTMLDivElement: align†
<dl>	HTMLDListElement: [boolean] compact†
<fieldset>	HTMLFieldSetElement: [readonly HTMLFormElement] form
<font>	HTMLFontElement†: color†, face†, size†
<form>	HTMLFormElement*: [readonly HTMLCollection] elements, [readonly long] length, name, acceptCharset, action, enctype, method, target, submit( ), reset( )
<frame>	HTMLFrameElement: frameBorder, longDesc, marginHeight, marginWidth, name, [boolean] noResize, scrolling, src, [readonly Document] contentDocument‡
<frameset>	HTMLFrameSetElement: cols, rows
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	HTMLHeadingElement: align†
<head>	HTMLHeadElement: profile
<hr>	HTMLHRElement: align†, [boolean] noShade†, size†, width†
<html>	HTMLHtmlElement: version†
<iframe>	HTMLIFrameElement: align†, frameBorder, height, longDesc, marginHeight, marginWidth, name, scrolling, src, width, [readonly Document] contentDocument‡
<img>	HTMLImageElement: align†, alt, [long] border†, [long] height, [long] hspace†, [boolean] isMap, longDesc, name, src, useMap, [long] vspace†, [long] width
<input>	HTMLInputElement*: defaultValue, [boolean] defaultChecked, [readonly HTMLFormElement] form, accept, accessKey, align†, alt, [boolean] checked, [boolean] disabled, [long] maxLength, name, [boolean] readOnly, size, src, [long] tabIndex, type, useMap, value, blur( ), focus( ), select( ), click( )
<ins>	See <del>
<isindex>	HTMLIsIndexElement†: [readonly HTMLFormElement] form, prompt†
<label>	HTMLLabelElement: [readonly HTMLFormElement] form, accessKey, htmlFor
<legend>	HTMLLegendElement: [readonly HTMLFormElement] form, accessKey, align†
	HTMLLIElement: type†, [long] value†
<link>	HTMLLinkElement: [boolean] disabled, charset, href, hreflang, media, rel, rev, target, type
<map>	HTMLMapElement: [readonly HTMLCollection of HTMLAreaElement] areas, name
<menu>	HTMLMenuElement†: [boolean] compact†
<meta>	HTMLMetaElement: content, httpEquiv, name, scheme
<object>	HTMLObjectElement: code, align†, archive, border†, codeBase, codeType, data, [boolean] declare, height, hspace†, name, standby, [long] tabIndex, type, useMap, vspace†, width, [readonly Document] contentDocument‡
	HTMLOListElement: [boolean] compact†, [long] start†, type†

HTML tag	DOM interface, properties, and methods
<optgroup>	HTMLOptGroupElement: [boolean] disabled, label
<option>	HTMLOptionElement*: [readonly HTMLFormElement] form, [boolean] defaultSelected, [readonly] text, [readonly long] index, [boolean] disabled, label, [boolean] selected, value
<p>	HTMLParagraphElement: align†
<param>	HTMLParamElement: name, type, value, valueType
<pre>	HTMLPreElement: [long] width†
<q>	See <blockquote>
<script>	HTMLScriptElement: text, htmlFor, event, charset, [boolean] defer, src, type
<select>	HTMLSelectElement*: [readonly] type, [long] selectedIndex, value, [readonly long] length, [readonly HTMLFormElement] form, [readonly HTMLCollection of HTMLOptionElement] options, [boolean] disabled, [boolean] multiple, name, [long] size, [long] tabIndex, add( ), remove( ), blur( ), focus( )
<style>	HTMLStyleElement: [boolean] disabled, media, type
<table>	HTMLTableElement*: [HTMLTableCaptionElement] caption, [HTMLTableSectionElement] tHead, [HTMLTableSectionElement] tFoot, [readonly HTMLCollection of HTMLTableRowElement] rows, [readonly HTMLCollection of HTMLTableSectionElement] tBodies, align†, bgColor†, border, cellPadding, cellSpacing, frame, rules, summary, width, createTHead( ), deleteTHead( ), createTFoot( ), deleteTFoot( ), createCaption( ), deleteCaption( ), insertRow( ), deleteRow( )
<tbody>, <tfoot>, <thead>	HTMLTableSectionElement*: align, ch, chOff, vAlign, [readonly HTMLCollection of HTMLTableRowElement] rows, insertRow( ), deleteRow( )
<td>, <th>	HTMLTableCellElement*: [readonly long] cellIndex, abbr, align, axis, bgColor†, ch, chOff, [long] colSpan, headers, height†, [boolean] noWrap†, [long] rowSpan, scope, vAlign, width†
<textarea>	HTMLTextAreaElement*: defaultValue, [readonly HTMLFormElement] form, accessKey, [long] cols, [boolean] disabled, name, [boolean] readOnly, [long] rows, [long] tabIndex, [readonly] type, value, blur( ), focus( ), select( )
<tfoot>	See <tbody>
<th>	See <td>
<thead>	See <tbody>
<title>	HTMLTitleElement: text
<tr>	HTMLTableRowElement*: [readonly long] rowIndex, [readonly long] sectionRowIndex, [readonly HTMLCollection of HTMLTableCellElement] cells, align, bgColor†, ch, chOff, vAlign, insertCell( ), deleteCell( )
	HTMLUListElement: [boolean] compact†, type†

* Indicates interfaces documented in this book.
† Indicates deprecated elements and attributes.
‡ Indicates attributes added in HTML DOM Level 2 working draft.

See Also

HTMLElement in the client-side reference section

Type of: HTMLDocument.body

Passed to: HTMLSelectElement.add()

Returned by: HTMLTableElement.createCaption(), HTMLTableElement.createTFoot(), HTMLTableElement.createTHead(), HTMLTableElement.insertRow(), HTMLTableRowElement.insertCell(), HTMLTableSectionElement.insertRow()

HTMLFormElement

DOM Level 1 HTML

a <form> in an HTML document

Node → Element → HTMLElement → HTMLFormElement

Properties

`readonly HTMLCollection elements`
: An array (HTMLCollection) of all elements in the form.

`readonly long length`
: The number of form elements in the form. This is the same value as `elements.length`.

In addition to the properties above, HTMLFormElement also defines the properties in the following table, which correspond directly to HTML attributes.

Property	Attribute	Description
`String acceptCharset`	`acceptcharset`	Character sets the server can accept
`String action`	`action`	URL of the form handler
`String enctype`	`enctype`	Encoding of the form
`String method`	`method`	HTTP method used for form submission
`String name`	`name`	Name of the form
`String target`	`target`	Frame or window name for form submission results

Methods

`reset()`
: Resets all form elements to their default values.

`submit()`
: Submits the form.

Description

This interface represents a `<form>` element in an HTML document. The `elements` property is an HTMLCollection that provides convenient access to all elements of the form. The `submit()` and `reset()` methods allow a form to be submitted or reset under program control.

See the Form object in the client-side reference section for more details.

See Also

Form object in the client-side reference section

Type of: HTMLButtonElement.form, HTMLFieldSetElement.form, HTMLInputElement.form, HTMLIsIndexElement.form, HTMLLabelElement.form, HTMLLegendElement.form, HTMLObjectElement.form, HTMLOptionElement.form, HTMLSelectElement.form, HTMLTextAreaElement.form

HTMLFormElement.reset( )

DOM Level 1 HTML

reset the elements of a form to their default values

Synopsis

```
void reset( );
```

Description

This method resets each of the elements of a form to its default value. The results of calling this method are like the results of a user clicking on a **Reset** button, except that the `onreset` event handler of the form is not invoked.

See Also

Form.reset( ) in the client-side reference section

HTMLFormElement.submit( )

DOM Level 1 HTML

submit a form

Synopsis

```
void submit( );
```

Description

This method submits the values of the form elements to the form handler specified by the form's `action` property. It submits a form in the same way that a user's clicking on a **Submit** button does, except that the `onsubmit` event handler of the form is not triggered.

See Also

Form.submit( ) in the client-side reference section

HTMLInputElement

DOM Level 1 HTML

an input element in an HTML form

Node → Element → HTMLElement → HTMLInputElement

Properties

`String accept`
: A comma-separated list of MIME types that specify the types of files that may be uploaded when this is a FileUpload element. Mirrors the `accept` attribute.

`String accessKey`
: The keyboard shortcut (which must be a single character) that a browser may use to transfer keyboard focus to this input element. Mirrors the `accesskey` attribute.

`deprecated String align`
: The vertical alignment of this element with respect to the surrounding text, or the left or right float for the element. Mirrors the `align` attribute.

`String alt`
: Alternate text to be displayed by browsers that cannot render this input element. Particularly useful when `type` is `image`. Mirrors the `alt` attribute.

`boolean checked`
: For Radio and Checkbox input elements, specifies whether the element is "checked" or not. Setting this property changes the visual appearance of the input element. Mirrors the `checked` attribute.

`boolean defaultChecked`
: For Radio and Checkbox elements, holds the initial value of the `checked` attribute as it appears in the document source. When the form is reset, the `checked` property is restored to the value of this property. Changing the value of this property changes the value of the `checked` property and the current checked state of the element.

`String defaultValue`
: For Text, Password, and FileUpload elements, holds the initial value displayed by the element. When the form is reset, the element is restored to this value. Changing the value of this property also changes the `value` property and the currently displayed value.

`boolean disabled`
: If `true`, the input element is disabled and is unavailable for user input. Mirrors the `disabled` attribute.

`readonly HTMLFormElement form`
: The HTMLFormElement object representing the `<form>` element that contains this input element, or `null` if the input element is not within a form.

`long maxLength`
: For Text or Password elements, specifies the maximum number of characters that the user will be allowed to enter. Note that this is not the same as the `size` property. Mirrors the `maxlength` attribute.

`String name`
: The name of the input element, as specified by the `name` attribute.

`boolean readOnly`
: If `true`, and this is a Text or Password element, the user is not allowed to enter text into the element. Mirrors the `readonly` attribute.

`String size`
: For Text and Password elements, specifies the width of the element in characters. Mirrors the `size` attribute. See also `maxLength`.

`String src`
: For input elements with a `type` of `image`, specifies the URL of the image to be displayed. Mirrors the `src` attribute.

`long tabIndex`
: The position of this input element in the tabbing order. Mirrors the `tabindex` attribute.

`String type`
: The type of the input element. The various types and their meanings are listed in the table in the "Description" section. Mirrors the `type` attribute.

`String useMap`
: For elements with a `type` of `image`, specifies the name of a `<map>` element that provides a client-side image map for the element.

`String value`
: The value that is passed to the server-side script when the form is submitted. For Text, Password, and FileUpload elements, this property is the text contained by the input element. For Button, Submit, and Reset elements, this is the text that appears in the button. For security reasons, the `value` property of FileUpload elements may be read-only. Similarly, the value returned by this property for Password elements may not contain the user's actual input.

Methods

`blur( )`
: Takes keyboard focus away from the element.

`click( )`
: If this input element is a Button, a Checkbox, or a Radio, Submit, or Reset button, this method simulates a mouse-click on the element.

`focus( )`
: Transfers keyboard focus to this input element.

`select( )`
: If this input element is a Text, Password, or FileUpload element, this method selects the text displayed by the element. In many browsers, this means that when the user next enters a character, the selected text will be deleted and replaced with the newly typed character.

Description

This interface represents an `<input>` element that defines an HTML input element (typically in an HTML form). An HTMLInputElement can represent various types of input elements, depending on the value of its `type` property. The allowed values for this property and their meanings are shown in the following table.

Type	Input element type
`button`	Push button
`checkbox`	Checkbox element
`file`	FileUpload element
`hidden`	Hidden element
`image`	Graphical Submit button
`password`	Masked-text entry field for passwords
`radio`	Mutually exclusive Radio button

Type	Input element type
`reset`	Reset button
`text` (default value)	Single-line text entry field
`submit`	Submit button

See Chapter 15 for more information about HTML forms and form elements. Note also that each distinct type of form input element has its own reference page in the client-side reference section of this book.

See Also

HTMLFormElement, HTMLOptionElement, HTMLSelectElement, HTMLTextAreaElement; Chapter 15; the Input object in the client-side reference section, and also its subclasses (Button, Checkbox, FileUpload, Hidden, Password, Radio, Reset, Submit, and Text)

HTMLInputElement.blur( )

DOM Level 1 HTML

take keyboard focus away from this element

Synopsis

```
void blur( );
```

Description

This method takes keyboard focus away from this form element.

HTMLInputElement.click( )

DOM Level 1 HTML

simulate a mouse-click on a form element

Synopsis

```
void click( );
```

Description

This method simulates a mouse-click on a Button, Checkbox, Radio, Reset, or Submit form element. It does not trigger the `onclick` event handler for the input element.

When called on Button elements, it may (or may not) produce the visual appearance of a button-click, but it has no other effect since it does not trigger the onclick event handler for the button. For Checkbox elements, it toggles the `checked` property. It makes unchecked Radio elements become checked but leaves checked elements alone. When called on Reset and Submit elements, the `click( )` method causes the form to be reset or submitted.

HTMLInputElement.focus( )

DOM Level 1 HTML

give keyboard focus to this element

Synopsis

```
void focus( );
```

Description

This method transfers keyboard focus to this element so the user can interact with it without first clicking on it.

HTMLInputElement.select( )

DOM Level 1 HTML

select the contents of a Text element

Synopsis

```
void select( );
```

Description

This method selects any text displayed in Text, Password, and FileUpload elements. In most browsers, this means that the user's next keystroke will replace the current text rather than being appended to it.

HTMLOptionElement

DOM Level 1 HTML

an <option> in an HTML form

Node → Element → HTMLElement → HTMLOptionElement

Properties

`boolean defaultSelected`
: The initial value of the `selected` attribute of the `<option>` element. If the form is reset, the `selected` property is reset to the value of this property. Setting this property also sets the value of the `selected` property.

`boolean disabled`
: If `true`, this option is disabled and the user is not allowed to select it. Mirrors the `disabled` attribute.

`readonly HTMLFormElement form`
: A reference to the `<form>` element that contains this element.

`readonly long index`
: The position of this `<option>` element within the `<select>` element that contains it.

`String label`
: The text to be displayed for the option. Mirrors the `label` attribute. If this property is not specified, the plain-text content of the `<option>` element is used instead.

`boolean selected`
: The current state of this option: if `true`, the option is selected. The initial value of this property comes from the `selected` attribute.

`readonly String text`
: The plain text contained within the `<option>` element. This text appears as the label for the option.

`String value`
: The value submitted with the form if this option is selected when form submission occurs. Mirrors the `value` attribute.

Description

This interface describes an `<option>` element within a `<select>` element.

See Also

HTMLFormElement, HTMLInputElement, HTMLSelectElement; Option and Select objects in the client-side reference section; Chapter 15

HTMLSelectElement

DOM Level 1 HTML

a <select> element in an HTML form

Node → Element → HTMLElement → HTMLSelectElement

Properties

`boolean disabled`
: If `true`, the `<select>` element is disabled and the user may not interact with it. Mirrors the `disabled` attribute.

`readonly HTMLFormElement form`
: The `<form>` element that contains this one.

`readonly long length`
: The number of `<option>` elements contained by this `<select>` element. Same as `options.length`.

`boolean multiple`
: If `true`, the `<select>` element allows multiple options to be selected. Otherwise, the selections are mutually exclusive and only one may be selected at a time. Mirrors the `multiple` attribute.

`String name`
: The name of this form element. Mirrors the `name` attribute.

`readonly HTMLCollection options`
: An array (HTMLCollection) of HTMLOptionElement objects that represent the `<option>` elements contained in this `<select>` element, in the order in which they appear.

`long selectedIndex`
: The position of the selected option in the `options` array. If no options are selected, this property is –1. If multiple options are selected, this property returns the index of the first selected option.

`long size`
: The number of options to display at once. If this property is 1, the `<select>` element will typically be displayed using a drop-down menu or list. If it is greater than 1, the `<select>` is typically displayed using a fixed-size list control, with a scrollbar if necessary. Mirrors the `size` attribute.

`long tabIndex`
: The position of this element in the tabbing order. Mirrors the `tabindex` attribute.

`readonly String type`
: If `multiple` is `true`, this property is "select-multiple". Otherwise, it is "select-one".

Methods

`add( )`
: Inserts a new HTMLOptionElement into the `options` array, either by appending it at the end of the array or by inserting it before another specified option.

`blur( )`
: Takes keyboard focus away.

`focus( )`
: Transfers keyboard focus to this element.

`remove( )`
: Removes the `<option>` element at the specified position.

Description

This interface represents a `<select>` element in an HTML form. The `options` property provides convenient access to the set of `<option>` elements it contains, and the `add( )` and `remove( )` methods provide an easy way to modify the set of options.

See Also

HTMLFormElement, HTMLOptionElement; Option and Select objects in the client-side reference section; Chapter 15

HTMLSelectElement.add( )

DOM Level 1 HTML

insert an <option> element

Synopsis

```
void add(HTMLElement element,
         HTMLElement before)
    throws DOMException;
```

Arguments

element
: The HTMLOptionElement to be added.

before
: The element of the `options` array before which the new *element* should be added. If this argument is `null`, *element* is appended at the end of the `options` array.

Throws

This method throws a DOMException with a code of NOT_FOUND_ERR if the *before* argument specifies an object that is not a member of the options array.

Description

This method adds a new <option> element to this <select> element. *element* is an HTMLOptionElement that represents the <option> element to be added. *before* specifies the HTMLOptionElement before which *element* is to be added. If *before* is part of an *OPTGROUP*, *element* is always inserted as part of that same group. If *before* is null, *element* becomes the last child of the <select> element.

See Also

Select object in the client-side reference section

HTMLSelectElement.blur( )

DOM Level 1 HTML

take keyboard focus away from this element

Synopsis

```
void blur( );
```

Description

This method takes keyboard focus away from this element.

HTMLSelectElement.focus( )

DOM Level 1 HTML

give keyboard focus to this element

Synopsis

```
void focus( );
```

Description

This method transfers keyboard focus to this <select> element so the user can interact with it using the keyboard instead of the mouse.

HTMLSelectElement.remove( )

DOM Level 1 HTML

remove an <option>

Synopsis

```
void remove(long index);
```

Arguments

index

The position within the options array of the <option> element to be removed.

Description

This method removes the <option> element at the specified position in the options array. If the specified *index* is less than zero or greater than or equal to the number of options, the remove( ) method ignores it and does nothing.

See Also

Select object in the client-side reference section

HTMLTableCaptionElement

DOM Level 1 HTML

a <caption> in an HTML table — Node → Element → HTMLElement → HTMLTableCaptionElement

Properties

deprecated String align
: The horizontal alignment of the caption with respect to the table. The value of the align attribute. Deprecated in favor of CSS styles.

Description

A <caption> element in an HTML table.

See Also

Type of: HTMLTableElement.caption

HTMLTableCellElement

DOM Level 1 HTML

a <td> or <th> cell in an HTML table — Node → Element → HTMLElement → HTMLTableCellElement

Properties

readonly long cellIndex
: The position of this cell within its row.

In addition to the cellIndex property, this interface defines the properties in the following table, which correspond directly to the HTML attributes of the <td> and <th> elements.

Property	Attribute	Description
String abbr	abbr	See HTML specification
String align	align	Horizontal alignment of cell
String axis	axis	See HTML specification
deprecated String bgColor	bgcolor	Background color of cell
String ch	char	Alignment character
String chOff	choff	Alignment character offset
long colSpan	colspan	Columns spanned by cell
String headers	headers	id values for headers for this cell
deprecated String height	height	Cell height in pixels

Property	Attribute	Description
deprecated boolean noWrap	nowrap	Don't word-wrap cell
long rowSpan	rowspan	Rows spanned by cell
String scope	scope	Scope of this header cell
String vAlign	valign	Vertical alignment of cell
deprecated String width	width	Cell width in pixels

Description

This interface represents <td> and <th> elements in HTML tables.

HTMLTableColElement

DOM Level 1 HTML

a <col> or <colgroup> in an HTML table — Node → Element → HTMLElement → HTMLTableColElement

Properties

This interface defines the properties in the following table, each of which corresponds to an HTML attribute of a <col> or <colgroup> element.

Property	Attribute	Description
String align	align	Default horizontal alignment
String ch	char	Default alignment character
String chOff	choff	Default alignment offset
long span	span	Number of columns represented by this element
String vAlign	valign	Default vertical alignment
String width	width	Width of the column(s)

Description

This interface represents a <col> or <colgroup> element in an HTML table.

HTMLTableElement

DOM Level 1 HTML

a <table> in an HTML document — Node → Element → HTMLElement → HTMLTableElement

Properties

`HTMLTableCaptionElement caption`
: A reference to the <caption> element for the table, or `null` if there is none.

`readonly HTMLCollection rows`
: An array (HTMLCollection) of HTMLTableRowElement objects that represent all the rows in the table. This includes all rows defined within <thead>, <tfoot>, and <tbody> tags.

`readonly HTMLCollection tBodies`
: An array (HTMLCollection) of HTMLTableSectionElement objects that represent all the <tbody> sections in this table.

`HTMLTableSectionElement tFoot`
: The `<tfoot>` element of the table, or `null` if there is none.

`HTMLTableSectionElement tHead`
: The `<thead>` element of the table, or `null` if there is none.

In addition to the properties just listed, this interface defines the properties in the following table to represent the HTML attributes of the `<table>` element.

Property	Attribute	Description
`deprecated String align`	`align`	Horizontal alignment of table in document
`deprecated String bgColor`	`bgcolor`	Table background color
`String border`	`border`	Width of border around table
`String cellPadding`	`cellpadding`	Space between cell contents and border
`String cellSpacing`	`cellspacing`	Space between cell borders
`String frame`	`frame`	Which table borders to draw
`String rules`	`rules`	Where to draw lines within the table
`String summary`	`summary`	Summary description of table
`String width`	`width`	Table width

Methods

`createCaption( )`
: Returns the existing `<caption>` for the table, or creates (and inserts) a new one if none already exists.

`createTFoot( )`
: Returns the existing `<tfoot>` element for the table, or creates (and inserts) a new one if none already exists.

`createTHead( )`
: Returns the existing `<thead>` element for the table, or creates (and inserts) a new one if none already exists.

`deleteCaption( )`
: Deletes the `<caption>` element from the table, if it has one.

`deleteRow( )`
: Deletes the row at the specified position in the table.

`deleteTFoot( )`
: Deletes the `<tfoot>` element from the table, if it has one.

`deleteTHead( )`
: Deletes the `<thead>` element from the table, if one exists.

`insertRow( )`
: Inserts a new, empty `<tr>` element into the table at the specified position.

Description

This interface represents an HTML `<table>` element and defines a number of convenience properties and methods for querying and modifying various sections of the table. These

methods and properties make it easier to work with tables, but they could also be duplicated with core DOM methods.

See Also

HTMLTableCaptionElement, HTMLTableCellElement, HTMLTableColElement, HTMLTableRowElement, HTMLTableSectionElement

HTMLTableElement.createCaption( ) DOM Level 1 HTML

get or create a <caption>

Synopsis

```
HTMLElement createCaption( );
```

Returns

An HTMLTableCaptionElement object representing the <caption> element for this table. If the table already has a caption, this method simply returns it. If the table does not have an existing <caption>, this method creates a new (empty) one and inserts it into the table before returning it.

HTMLTableElement.createTFoot( ) DOM Level 1 HTML

get or create a <tfoot>

Synopsis

```
HTMLElement createTFoot( );
```

Returns

An HTMLTableSectionElement representing the <tfoot> element for this table. If the table already has a footer, this method simply returns it. If the table does not have an existing footer, this method creates a new (empty) <tfoot> element and inserts it into the table before returning it.

HTMLTableElement.createTHead( ) DOM Level 1 HTML

get or create a <thead>

Synopsis

```
HTMLElement createTHead( );
```

Returns

An HTMLTableSectionElement representing the <thead> element for this table. If the table already has a header, this method simply returns it. If the table does not have an existing header, this method creates a new (empty) <thead> element and inserts it into the table before returning it.

HTMLTableElement.deleteCaption( )

DOM Level 1 HTML

delete the <caption> of a table

Synopsis

```
void deleteCaption( );
```

Description

If this table has a <caption> element, this method removes it from the document tree. Otherwise, it does nothing.

HTMLTableElement.deleteRow( )

DOM Level 1 HTML

delete a row of a table

Synopsis

```
void deleteRow(long index)
    throws DOMException;
```

Arguments

index
: Specifies the position within the table of the row to be deleted.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is less than zero or is greater than or equal to the number of rows in the table.

Description

This method deletes the row at the specified position from the table. Rows are numbered in the order in which they appear in the document source. Rows in <thead> and <tfoot> sections are numbered along with all other rows in the table.

See Also

HTMLTableSectionElement.deleteRow( )

HTMLTableElement.deleteTFoot( )

DOM Level 1 HTML

delete the <tfoot> of a table

Synopsis

```
void deleteTFoot( );
```

Description

If this table has a <tfoot> element, this method removes it from the document tree. If the table has no footer, this method does nothing.

HTMLTableElement.deleteTHead( ) DOM Level 1 HTML

delete the <thead> of a table

Synopsis

```
void deleteTHead( );
```

Description

If this table has a <thead> element, this method deletes it; otherwise, it does nothing.

HTMLTableElement.insertRow( ) DOM Level 1 HTML

add a new, empty row to the table

Synopsis

```
HTMLElement insertRow(long index)
    throws DOMException;
```

Arguments

index
: The position at which the new row is to be inserted.

Returns

An HTMLTableRowElement that represents the newly inserted row.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is less than zero or greater than the number of rows in the table.

Description

This method creates a new HTMLTableRowElement representing a <tr> tag and inserts it into the table at the specified position.

The new row is inserted in the same section and immediately before the existing row at the position specified by *index*. If *index* is equal to the number of rows in the table, the new row is appended to the last section of the table. If the table is initially empty, the new row is inserted into a new <tbody> section that is itself inserted into the table.

You can use the convenience method HTMLTableRowElement.insertCell( ) to add content to the newly created row.

See Also

HTMLTableSectionElement.insertRow( )

HTMLTableRowElement

DOM Level 1 HTML

a <tr> element in an HTML table

Node → Element → HTMLElement → HTMLTableRowElement

Properties

`readonly HTMLCollection cells`
: An array (HTMLCollection) of HTMLTableCellElement objects representing the cells in this row.

`readonly long rowIndex`
: The position of this row in the table.

`readonly long sectionRowIndex`
: The position of this row within its section (i.e., within its <thead>, <tbody>, or <tfoot> element).

In addition to the properties just listed, this interface also defines the properties in the following table, which correspond to the HTML attributes of the <tr> element.

Property	Attribute	Description
`String align`	`align`	Default horizontal alignment of cells in this row
`deprecated String bgColor`	`bgcolor`	Background color of this row
`String ch`	`char`	Alignment character for cells in this row
`String chOff`	`choff`	Alignment character offset for cells in this row
`String vAlign`	`valign`	Default vertical alignment for cells in this row

Methods

`deleteCell( )`
: Deletes the specified cell from this row.

`insertCell( )`
: Inserts an empty <td> element into this row at the specified position.

Description

This interface represents a row in an HTML table.

HTMLTableRowElement.deleteCell( )

DOM Level 1 HTML

delete a cell in a table row

Synopsis

```
void deleteCell(long index)
    throws DOMException;
```

Arguments

index
: The position in the row of the cell to delete.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is less than zero or is greater than or equal to the number of cells in the row.

Description

This method deletes the cell at the specified position in the table row.

HTMLTableRowElement.insertCell( )

DOM Level 1 HTML

insert a new, empty <td> element into a table row

Synopsis

```
HTMLElement insertCell(long index)
    throws DOMException;
```

Arguments

index
: The position at which the new cell is to be inserted.

Returns

An HTMLTableCellElement object that represents the newly created and inserted <td> element.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is less than zero or is greater than the number of cells in the row.

Description

This method creates a new <td> element and inserts it into the row at the specified position. The new cell is inserted immediately before the cell that is currently at the position specified by *index*. If *index* is equal to the number of cells in the row, the new cell is appended at the end of the row.

Note that this convenience method inserts <td> data cells only. If you need to add a header cell into a row, you must create and insert the <th> element using Document.createElement( ) and Node.insertBefore( ) or related methods.

HTMLTableSectionElement

DOM Level 1 HTML

a header, footer, or body section of a table

Node → Element → HTMLElement → HTMLTableSectionElement

Properties

readonly HTMLCollection rows
: An array (HTMLCollection) of HTMLTableRowElement objects representing the rows in this section of the table.

In addition to the rows property, this interface defines the properties in the following table, which represent the attributes of the underlying HTML element.

Property	Attribute	Description
String align	align	Default horizontal alignment of cells in this section of the table
String ch	char	Default alignment character for cells in this section
String chOff	choff	Default alignment offset for cells in this section
String vAlign	valign	Default vertical alignment for cells in this section

Methods

deleteRow( )
: Deletes the numbered row from this section.

insertRow( )
: Inserts an empty row into this section at the specified position.

Description

This interface represents a <tbody>, <thead>, or <tfoot> section of an HTML table.

See Also

Type of: HTMLTableElement.tFoot, HTMLTableElement.tHead

HTMLTableSectionElement.deleteRow( )

DOM Level 1 HTML

delete a row within a table section

Synopsis

```
void deleteRow(long index)
    throws DOMException;
```

Arguments

index
: The position of the row within this section.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is less than zero or is greater than or equal to the number of rows in this section.

Description

This method deletes the row at the specified position within this section. Note that for this method *index* specifies a row's position within its section, not within the entire table.

See Also

HTMLTableElement.deleteRow( )

HTMLTableSectionElement.insertRow( )

DOM Level 1 HTML

insert a new, empty row into this table section

Synopsis

```
HTMLElement insertRow(long index)
    throws DOMException;
```

Arguments

index
: The position within the section at which the new row is to be inserted.

Returns

An HTMLTableRowElement that represents the newly created and inserted <tr> element.

Throws

This method throws a DOMException with a code of INDEX_SIZE_ERR if *index* is less than zero or is greater than the number of rows in this section.

Description

This method creates a new <tr> element and inserts it into this table section at the specified position. If *index* equals the number of rows currently in the section, the new row is appended at the end of the section. Otherwise, the new row is inserted immediately before the row that is currently at the position specified by *index*. Note that for this method, *index* specifies a row position within a single table section, not within the entire table.

See Also

HTMLTableElement.insertRow( )

HTMLTextAreaElement

DOM Level 1 HTML

a <textarea> element in an HTML form

Node → Element → HTMLElement → HTMLTextAreaElement

Properties

`String accessKey`
: A keyboard shortcut (a single character) that the web browser can use to transfer keyboard focus to this element. Mirrors the accesskey attribute.

`long cols`
: The width of this element in character columns. Mirrors the cols attribute.

`String defaultValue`
: The initial content of the text area. When the form is reset, the text area is restored to this value. Setting this property changes the displayed text in the text area.

`boolean disabled`
: If true, this element is disabled and the user cannot interact with it. Mirrors the disabled attribute.

`readonly HTMLFormElement form`
: The HTMLFormElement that represents the `<form>` element containing this text area, or `null` if this element is not inside a form.

`String name`
: The name of this `<textarea>` element, as specified by the `name` attribute.

`boolean readOnly`
: If `true`, this element is read-only and the user cannot edit any of the displayed text. Mirrors the `readonly` attribute.

`long rows`
: The height of the text area in text rows. Mirrors the `rows` attribute.

`long tabIndex`
: The position of this element in the tabbing order. Mirrors the `tabindex` attribute.

`readonly String type`
: The type of this element, for compatibility with HTMLInputElement objects. This property always has the value "textarea".

`String value`
: The text currently displayed in the text area.

Methods

`blur( )`	Takes keyboard focus away from this element.
`focus( )`	Transfers keyboard focus to this element.
`select( )`	Selects the entire contents of the text area.

Description

This interface represents a `<textarea>` element that creates a multiline text-input field in an HTML form. The initial contents of the text area are specified between the `<textarea>` and `</textarea>` tags. The user may edit this value and query and set the text with the `value` property (or by modifying the Text node child of this element).

See Also

HTMLFormElement, HTMLInputElement; Textarea in the client-side reference section; Chapter 15

HTMLTextAreaElement.blur( )

DOM Level 1 HTML

take keyboard focus away from this element

Synopsis

```
void blur( );
```

Description

This method takes keyboard focus away from this element.

HTMLTextAreaElement.focus( )

DOM Level 1 HTML

give keyboard focus to this element

Synopsis

```
void focus( );
```

Description

This method transfers keyboard focus to this element so the user can edit the displayed text without having to first click on the text area.

HTMLTextAreaElement.select( )

DOM Level 1 HTML

select the text in this element

Synopsis

```
void select( );
```

Description

This method selects all the text displayed by this `<textarea>` element. In most browsers, this means that the text is highlighted and that new text entered by the user will replace the highlighted text instead of being appended to it.

LinkStyle

DOM Level 2 StyleSheets

a style sheet associated with a node

Properties

`readonly StyleSheet sheet`
: The StyleSheet object associated with this node.

Description

In DOM implementations that support the StyleSheets module, this interface is implemented by any Document node that links to a style sheet or defines an inline style sheet. The `sheet` property then provides a way to obtain the StyleSheet object associated with the node.

In HTML documents, the `<style>` and `<link>` elements implement this interface. Those elements are represented by the HTMLStyleElement and HTMLLinkElement interfaces, which do not have their own entries in this reference. See "HTMLElement" for more information about those interfaces.

In XML documents, style sheets are included with a processing instruction. See "ProcessingInstruction" for more information.

See Also

HTMLElement, ProcessingInstruction

MediaList

DOM Level 2 StyleSheets

a style sheet's list of media types

Properties

`readonly unsigned long length`
: The length of the array; the number of media types in the list.

`String mediaText`
: A comma-separated text representation of the complete media list. Setting this property may throw a DOMException with a `code` of `SYNTAX_ERR` if the new value contains a syntax err, or a `code` of `NO_MODIFICATION_ALLOWED_EXCEPTION` if the media list is read-only.

Methods

`appendMedium( )`
: Adds a new media type to the end of the list.

`deleteMedium( )`
: Removes the specified media type from the list.

`item( )`
: Returns the media type at the specified position in the list, or `null` if the index is invalid. In JavaScript, you can also treat the MediaList object as an array and index it using normal square-bracket array notation instead of calling this method.

Description

This interface represents a list or array of media types for a style sheet. `length` specifies the number of elements in the list, and `item( )` allows a specific media type to be retrieved by position. `appendMedium( )` and `deleteMedium( )` allow entries to be appended to and deleted from the list. JavaScript allows a MediaList object to be treated as an array, and you can use square-bracket notation instead of calling `item( )`.

The HTML 4 standard defines the following media types (they are case-sensitive, and must be written in lowercase letters): `screen`, `tty`, `tv`, `projection`, `handheld`, `print`, `braile`, `aural`, and `all`. The `screen` type is most relevant to documents being displayed in web browsers on desktop or laptop computers. The `print` type is used for styles intended for printed documents.

See Also

Type of: StyleSheet.media

MediaList.appendMedium( )

DOM Level 2 StyleSheets

add a new media type to the list

Synopsis

```
void appendMedium(String newMedium)
    throws DOMException;
```

Arguments

newMedium

The name of the new media type to append. See the "MediaList" reference page for the set of valid media type names.

Throws

This method may throw a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if the media list is read-only, or `INVALID_CHARACTER_ERR` if the specified *newMedium* argument contains illegal characters.

Description

This method appends the specified *newMedium* to the end of the MediaList. If the MediaList already contains the specified media type, it is first removed from its current position and then appended at the end of the list.

MediaList.deleteMedium( )

DOM Level 2 StyleSheets

remove a media type from the list

Synopsis

```
void deleteMedium(String oldMedium)
    throws DOMException;
```

Arguments

oldMedium

The name of the media type to remove from the list. See the "MediaList" reference page for the set of valid media type names.

Throws

This method throws a DOMException with a `code` of `NOT_FOUND_ERR` if the list does not contain the specified *oldMedium* media type, or `NO_MODIFICATION_ALLOWED_ERR` if the media list is read-only.

Description

This method deletes the specified media type from this MediaList, or throws an exception if the list does not contain the specified type.

MediaList.item( )

DOM Level 2 StyleSheets

index an array of media types

Synopsis

```
String item(unsigned long index);
```

Arguments

index
: The position of the desired media type within the array.

Returns

The media type (a string) at the specified position within the MediaList, or `null` if *index* is negative or is greater than or equal to `length`. Note that in JavaScript, it is usually simpler to treat a MediaList object as an array and index it using square-bracket array notation instead of calling this method.

MouseEvent

DOM Level 2 Events

details about a mouse event

Event → UIEvent → MouseEvent

Properties

`readonly boolean altKey`
: Whether the **Alt** key was held down when the event occurred. Defined for all types of mouse events.

`readonly unsigned short button`
: Which mouse button changed state during a mousedown, mouseup, or click event. A value of 0 indicates the left button, a value of 2 indicates the right button, and a value of 1 indicates the middle mouse button. Note that this property is defined when a button changes state; it is not used to report whether a button is held down during a mousemove event, for example. Also, this property is not a bitmap: it cannot tell you if more than one button is held down.

  Netscape 6.0 and 6.01 use the values 1, 2, and 3 instead of 0, 1, and 2. This is fixed in Netscape 6.1.

`readonly long clientX, clientY`
: Numbers that specify the X and Y coordinates of the mouse pointer relative to the "client area," or browser window. Note that these coordinates do not take document scrolling into account; if an event occurs at the very top of the window, `clientY` is 0 regardless of how far down the document has been scrolled. These properties are defined for all types of mouse events.

`readonly boolean ctrlKey`
: Whether the **Ctrl** key was held down when the event occurred. Defined for all types of mouse events.

`readonly boolean metaKey`
: Whether the **Meta** key was held down when the event occurred. Defined for all types of mouse events.

`readonly EventTarget relatedTarget`
: Refers to a node that is related to the `target` node of the event. For mouseover events, it is the node the mouse left when it moved over the target. For mouseout events, it is the node the mouse entered when leaving the target. `relatedTarget` is undefined for other types of mouse events.

`readonly long screenX, screenY`
: Numbers that specify the X and Y coordinates of the mouse pointer relative to the upper-left corner of the user's monitor. These properties are defined for all types of mouse events.

`readonly boolean shiftKey`
: Whether the **Shift** key was held down when the event occurred. Defined for all types of mouse events.

Methods

`initMouseEvent( )`
: Initializes the properties of a newly created MouseEvent object.

Description

This interface defines the type of Event object that is passed to events of types click, mousedown, mousemove, mouseout, mouseover, and mouseup. Note that in addition to the properties listed here, this interface also inherits the properties of the UIEvent and Event interfaces.

See Also

Event, UIEvent; Chapter 19

MouseEvent.initMouseEvent( )

DOM Level 2 Events

initialize the properties of a MouseEvent object

Synopsis

```
void initMouseEvent(String typeArg,
                    boolean canBubbleArg,
                    boolean cancelableArg,
                    AbstractView viewArg,
                    long detailArg,
                    long screenXArg,
                    long screenYArg,
                    long clientXArg,
                    long clientYArg,
                    boolean ctrlKeyArg,
                    boolean altKeyArg,
                    boolean shiftKeyArg,
```

```
boolean metaKeyArg,
unsigned short buttonArg,
EventTarget relatedTargetArg);
```

Arguments

The many arguments to this method specify the initial values of the properties of this MouseEvent object, including the properties inherited from the Event and UIEvent interfaces. The name of each argument clearly indicates the property for which it specifies the value, so they are not listed individually here.

Description

This method initializes the various properties of a newly created MouseEvent object. It may be called only on a MouseEvent object created with `Document.createEvent()` and only before that MouseEvent is passed to `EventTarget.dispatchEvent()`.

MutationEvent

DOM Level 2 Events

details about a document change

Event → MutationEvent

Constants

The following constants represent the set of possible values for the `attrChange` property:

`unsigned short MODIFICATION = 1`
: An Attr node was modified.

`unsigned short ADDITION = 2`
: An Attr node was added.

`unsigned short REMOVAL = 3`
: An Attr node was removed.

Properties

`readonly unsigned short attrChange`
: How the attribute was changed, for DOMAttrModified events. The three possible values are defined in the "Constants" section.

`readonly String attrName`
: The name of the attribute that was changed for DOMAttrModified events.

`readonly String newValue`
: The new value of the Attr node for DOMAttrModified events, or the new text value of a Text, Comment, CDATASection, or ProcessingInstruction node for DOMCharacterDataModified events.

`readonly String prevValue`
: The previous value of an Attr node for DOMAttrModified events, or the previous value of a Text, Comment, CDATASection, or ProcessingInstruction node for DOMCharacterDataModified events.

`readonly Node relatedNode`
: The relevant Attr node for DOMAttrModified events, or the parent of the node that was inserted or removed for DOMNodeInserted and DOMNodeRemoved events.

Methods

`initMutationEvent( )`
: Initializes the properties of a newly created MutationEvent object.

Description

This interface defines the type of Event object that is passed to events of types listed here (note that none of these event types are cancelable with `Event.preventDefault( )`):

DOMAttrModified
: Generated when an attribute of a document element is added, removed, or changed. The target of the event is the element that contains the attribute, and the event bubbles up from there.

DOMCharacterDataModified
: Generated when the character data of a Text, Comment, CDATASection, or ProcessingInstruction node changes. The event target is the node that changed, and the event bubbles up from there.

DOMNodeInserted
: Generated after a node is added as a child of another node. The target of the event is the node that was inserted, and the event bubbles up the tree from there. The `relatedNode` property specifies the new parent of the inserted node. This event is not generated for any descendants of the inserted node.

DOMNodeInsertedIntoDocument
: Generated after a node is inserted into the document tree, as well as for nodes that are inserted directly into the tree and nodes that are indirectly inserted when an ancestor is inserted. The target of this event is the node that is inserted. Because events of this type may be targeted at every node in a subtree, they do not bubble.

DOMNodeRemoved
: Generated immediately before a node is removed from its parent. The target of the event is the node being removed, and the event bubbles up the document tree from there. The `relatedNode` property holds the parent node from which the node is being removed.

DOMNodeRemovedFromDocument
: Generated immediately before a node is removed from the document tree. Separate events are generated for the node that is directly removed and for each of its descendant nodes. The target of the event is the node that is about to be removed. Events of this type do not bubble.

DOMSubtreeModified
: Generated as a kind of summary event when a call to a DOM method causes multiple mutation events to be fired. The target of this event is the most deeply nested common ancestor of all changes that occurred in the document, and it bubbles up the document tree from that point. If you are not interested in the details of the changes but merely want to be notified which portions of the document have changed, you may prefer to register listeners for this type of event.

MutationEvent.initMutationEvent( ) DOM Level 2 Events

initialize the properties of a new MutationEvent

Synopsis

```
void initMutationEvent(String typeArg,
                       boolean canBubbleArg,
                       boolean cancelableArg,
                       Node relatedNodeArg,
                       String prevValueArg,
                       String newValueArg,
                       String attrNameArg,
                       unsigned short attrChangeArg);
```

Arguments

The various arguments to this method specify the initial values of the properties of this MutationEvent object, including the properties inherited from the Event interface. The name of each argument clearly indicates the property for which it specifies the value, so the arguments are not listed individually here.

Description

This method initializes the various properties of a newly created MutationEvent object. It may be called only on a MutationEvent object created with `Document.createEvent( )` and only before that MouseEvent is passed to `EventTarget.dispatchEvent( )`.

NamedNodeMap DOM Level 1 Core

a collection of nodes indexed by name or position

Properties

`readonly unsigned long length`
: The number of nodes in the array.

Methods

`getNamedItem( )`
: Looks up a named node.

`getNamedItemNS( )` *[DOM Level 2]*
: Looks up a node specified by namespace and name.

`item( )`
: Obtains the node at a specified position within the NamedNodeMap. In JavaScript, you can also do this by using the node position as an array index.

`removeNamedItem( )`
: Deletes a named node from the NamedNodeMap.

`removeNamedItemNS( )` *[DOM Level 2]*
: Deletes a node specified by name and namespace from the NamedNodeMap.

`setNamedItem( )`
: Adds a new node to (or replaces an existing node in) the NamedNodeMap. The `nodeName` property of the Node object is used as the name of the node.

`setNamedItemNS( )` *[DOM Level 2]*
: Adds a new node to (or replaces an existing node in) the NamedNodeMap. The `namespaceURI` and `localName` properties of the Node object are used as the node name.

Description

The NamedNodeMap interface defines a collection of nodes that may be looked up by their `nodeName` property or, for nodes that use namespaces, by their `namespaceURI` or `localName` properties.

The most notable use of the NamedNodeMap interface is the `attributes` property of the Node interface: a collection of Attr nodes that may be looked up by attribute name. Many of the methods of NamedNodeMap are similar to the methods of Element for manipulating attributes. Element attributes are usually most easily manipulated through the methods of the Element interface, and the NamedNodeMap interface is not commonly used.

NamedNodeMap objects are "live," which means that they immediately reflect any changes to the document tree. For example, if you obtain a NamedNodeMap that represents the attributes of an element and then add a new attribute to that element, the new attribute is available through the NamedNodeMap.

See Also

NodeList

Type of: DocumentType.entities, DocumentType.notations, Node.attributes

NamedNodeMap.getNamedItem( ) DOM Level 1 Core

look up a node by name

Synopsis

```
Node getNamedItem(String name);
```

Arguments

name
: The value of the `nodeName` property of the node to look up.

Returns

The named node, or `null` if no node with that name was found.

NamedNodeMap.getNamedItemNS( )

DOM Level 2 Core

look up a node by name and namespace

Synopsis

```
Node getNamedItemNS(String namespaceURI,
                    String localName);
```

Arguments

namespaceURI
: The `namespaceURI` property of the desired node, or `null` for no namespace.

localName
: The `localName` property of the local node.

Returns

The element of the NamedNodeMap that has the specified `namespaceURI` and `localName` properties, or `null` if there is no such node.

Description

`getNamedItemNS( )` looks up an element of a NamedNodeMap by namespace and local name. It is useful only with XML documents that use namespaces.

NamedNodeMap.item( )

DOM Level 1 Core

return an element of a NamedNodeMap by position

Synopsis

```
Node item(unsigned long index);
```

Arguments

index
: The position or index of the desired node.

Returns

The node at the specified position, or `null` if *index* is less than zero or greater than or equal to the length of the NamedNodeMap.

Description

This method returns a numbered element of a NamedNodeMap. In JavaScript, NamedNodeMap objects behave like read-only arrays, and you can use the node position as an array index within square brackets instead of calling this method.

Although the NamedNodeMap interface allows you to iterate through its nodes by position, it does not represent an ordered collection of nodes. Any changes made to the

NamedNodeMap (such as by removeNamedItem( ) or setNamedItem( )) may result in a complete reordering of the elements. Thus, you must not modify a NamedNodeMap while you are iterating through its elements.

See Also

NodeList

NamedNodeMap.removeNamedItem( ) DOM Level 1 Core

delete a node specified by name

Synopsis

```
Node removeNamedItem(String name)
    throws DOMException;
```

Arguments

name
: The nodeName property of the node to be deleted.

Returns

The node that was removed.

Throws

This method throws a DOMException with a code of NO_MODIFICATION_ALLOWED_ERR if the NamedNodeMap is read-only and does not allow deletions, or a code of NOT_FOUND_ERR if no node with the specified *name* exists in the NamedNodeMap.

Description

Deletes a named node from a NamedNodeMap. Note that if the NamedNodeMap represents the set of attributes for an Element, removing the Attr node for an attribute that was explicitly set in the document may cause the removed Attr to be automatically replaced by a new Attr node representing the default value (if any exists) of the attribute.

See Also

Element.removeAttribute( )

NamedNodeMap.removeNamedItemNS( ) DOM Level 2 Core

delete a node specified by namespace and name

Synopsis

```
Node removeNamedItemNS(String namespaceURI,
                       String localName)
    throws DOMException;
```

Arguments

namespaceURI
: The `namespaceURI` property of the node to be removed, or `null` for no namespace.

localName
: The `localName` property of the node to be removed.

Returns

The node that was removed.

Throws

This method throws exceptions for the same reason as `removeNamedItem( )`.

Description

This method works just like `removeNamedItem( )`, except that the node to be removed is specified by namespace and local name rather than just by name. This method is typically useful only with XML documents that use namespaces.

NamedNodeMap.setNamedItem( )

DOM Level 1 Core

add a node to or replace a node in a NamedNodeMap

Synopsis

```
Node setNamedItem(Node arg)
    throws DOMException;
```

Arguments

arg
: The node to be added to the NamedNodeMap.

Returns

The node that was replaced, or `null` if no node was replaced.

Throws

This method may throw a DOMException with one of the following `code` values:

`HIERARCHY_REQUEST_ERR`
: *arg* is a node of a type that is not suitable for this NamedNodeMap (e.g., is not an Attr node).

`INUSE_ATTRIBUTE_ERR`
: *arg* is an Attr node that is already associated with an element.

`NO_MODIFICATION_ALLOWED_ERR`
: The NamedNodeMap is read-only.

`WRONG_DOCUMENT_ERR`
: *arg* has a different `ownerDocument` than the document from which the NamedNodeMap was created.

Description

`setNamedItem( )` adds the specified node to a NamedNodeMap and allows it to be looked up using the value of the node's `nodeName` property. If the NamedNodeMap already contains a node with that name, that node is replaced and becomes the return value of the method.

See Also

Element.setAttribute( )

NamedNodeMap.setNamedItemNS( ) DOM Level 2 Core

add a node to a NamedNodeMap using namespaces

Synopsis

```
Node setNamedItemNS(Node arg)
    throws DOMException;
```

Arguments

arg
: The node to be added to the NamedNodeMap.

Returns

The node that was replaced, or `null` if no node was replaced.

Throws

This method throws exceptions for the same reasons as `setNamedItem( )`. It may also throw a DOMException with a `code` of `NOT_SUPPORTED_ERR` if it is called in an implementation that does not support XML documents or XML namespaces.

Description

This method works like `setNamedItem( )`, except that the node added to the NamedNodeMap can later be looked up by its `namespaceURI` and `localName` properties instead of by its `nodeName` property. This method is useful only with XML documents that use namespaces. Note that this method may be unsupported (i.e., may throw an exception) in implementations that do not support XML documents.

Node DOM Level 1 Core

a node in a document tree

Subinterfaces

Attr, CharacterData, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction

Also Implements

EventTarget
: If the DOM implementation supports the Events module, every node in the document tree also implements the EventTarget interface and may have event listeners registered on it. The methods defined by the EventTarget interface are not included here; see the "EventTarget" and "EventListener" reference pages for details.

Constants

All Node objects implement one of the subinterfaces listed above. Every Node object has a `nodeType` property that specifies which of the subinterfaces it implements. These constants are the legal values for that property; their names are self-explanatory. Note that these are static properties of the `Node()` constructor function; they are not properties of individual Node objects. Also note that they are not supported by Internet Explorer 4, 5, or 6.

```
Node.ELEMENT_NODE = 1;                // Element
Node.ATTRIBUTE_NODE = 2;              // Attr
Node.TEXT_NODE = 3;                   // Text
Node.CDATA_SECTION_NODE = 4;          // CDATASection
Node.ENTITY_REFERENCE_NODE = 5;       // EntityReference
Node.ENTITY_NODE = 6;                 // Entity
Node.PROCESSING_INSTRUCTION_NODE = 7; // ProcessingInstruction
Node.COMMENT_NODE = 8;                // Comment
Node.DOCUMENT_NODE = 9;               // Document
Node.DOCUMENT_TYPE_NODE = 10;         // DocumentType
Node.DOCUMENT_FRAGMENT_NODE = 11;     // DocumentFragment
Node.NOTATION_NODE = 12;              // Notation
```

Properties

`readonly NamedNodeMap attributes`
: If this node is an element, specifies the attributes of that element. `attributes` is a NamedNodeMap object that allows attributes to be queried by name or by number and returns them in the form of Attr objects. In practice, it is almost always easier to use the `getAttribute()` method of the Element interface to obtain an attribute value as a string. Note that the returned NamedNodeMap object is "live": any changes to the attributes of this element are immediately visible through it.

`readonly Node[] childNodes`
: Contains the child nodes of the current node. This property should never be `null`: for nodes with no children, `childNodes` is an array with `length` zero. This property is technically a NodeList object, but it behaves just like an array of Node objects. Note that the returned NodeList object is "live": any changes to this element's list of children are immediately visible through the NodeList.

`readonly Node firstChild`
: The first child of this node, or `null` if the node has no children.

`readonly Node lastChild`
: The last child of this node, or `null` if the node has no children.

`readonly String localName` *[DOM Level 2]*
: In XML documents that use namespaces, specifies the local part of the element or attribute name. This property is never used with HTML documents. See also the `namespaceURI` and `prefix` properties.

`readonly String namespaceURI` *[DOM Level 2]*
: In XML documents that use namespaces, specifies the URI of the namespace of an Element or Attribute node. This property is never used with HTML documents. See also the `localName` and `prefix` properties.

`readonly Node nextSibling`
: The sibling node that immediately follows this one in the `childNodes[]` array of the `parentNode`, or `null` if there is no such node.

`readonly String nodeName`
: The name of the node. For Element nodes, specifies the tag name of the element, which can also be retrieved with the `tagName` property of the Element interface. For other types of nodes, the value depends on the node type. See the upcoming table in the "Descriptions" section for details.

`readonly unsigned short nodeType`
: The type of the node; i.e., which subinterface the node implements. The legal values are defined by the previously listed constants. Since those constants are not supported by Internet Explorer, however, you may prefer to use hardcoded values instead of the constants. In HTML documents, the common values for this property are 1 for Element nodes, 3 for Text nodes, 8 for Comment nodes, and 9 for the single top-level Document node.

`String nodeValue`
: The value of a node. For Text nodes, holds the text content. For other node types, the value depends on the `nodeType`, as shown in the upcoming table.

`readonly Document ownerDocument`
: The Document object of which this node is a part. For Document nodes, this property is `null`.

`readonly Node parentNode`
: The parent node (or container node) of this node, or `null` if there is no parent. Note that Document and Attr nodes never have parent nodes. Also, nodes that have been removed from the document or are newly created and have not yet been inserted into the document tree have a `parentNode` of `null`.

`String prefix` *[DOM Level 2]*
: For XML documents that use namespaces, specifies the namespace prefix of an Element or Attribute node. This property is never used with HTML documents. See also the `localName` and `namespaceURL` properties. Setting this property can cause an exception if the new value contains illegal characters, is malformed, or does not match the `namespaceURI` property.

`readonly Node previousSibling`
: The sibling node that immediately precedes this one in the `childNodes[]` array of the `parentNode`, or `null` if there is no such node.

Methods

`appendChild( )`
: Adds a node to the document tree by appending it to the `childNodes[]` array of this node. If the node is already in the document tree, it is removed and then reinserted at its new position.

`cloneNode( )`
: Makes a copy of this node, or of the node and all its descendants.

`hasAttributes( )` *[DOM Level 2]*
: Returns `true` if this node is an Element and has any attributes.

`hasChildNodes( )`
: Returns `true` if this node has any children.

`insertBefore( )`
: Inserts a node into the document tree immediately before the specified child of this node. If the node being inserted is already in the tree, it is removed and reinserted at its new location.

`isSupported( )` *[DOM Level 2]*
: Returns `true` if the specified version number of a named feature is supported by this node.

`normalize( )`
: "Normalizes" all Text node descendants of this node by deleting empty Text nodes and merging adjacent Text nodes.

`removeChild( )`
: Removes (and returns) the specified child node from the document tree.

`replaceChild( )`
: Removes (and returns) the specified child node from the document tree, replacing it with another node.

Description

All objects in a document tree (including the Document object itself) implement the Node interface, which provides the fundamental properties and methods for traversing and manipulating the tree. The `parentNode` property and `childNodes[]` array allow you to move up and down the document tree. You can enumerate the children of a given node by looping through the elements of `childNodes[]` or by using the `firstChild` and `nextSibling` properties (or the `lastChild` and `previousSibling` properties, to loop backward). The `appendChild( )`, `insertBefore( )`, `removeChild( )`, and `replaceChild( )` methods allow you to modify the document tree by altering the children of a node.

Every object in a document tree implements both the Node interface and a more specialized interface, such as Element or Text. The `nodeType` property specifies which subinterface a node implements. You can use this property to test the type of a node before using properties or methods of the more specialized interface. For example:

```
var n;   // Holds the node we're working with
if (n.nodeType == 1) {        // Or compare to the constant Node.ELEMENT_NODE
    var tagname = n.tagName;  // If the node is an Element, this is the tag name
}
```

The nodeName and nodeValue properties specify additional information about a node, but their value depends on nodeType, as shown in the following table. Note that subinterfaces typically define specialized properties (such as the tagName property of Element nodes and the data property of Text nodes) for obtaining this information.

nodeType	nodeName	nodeValue
ELEMENT_NODE	The element's tag name	null
ATTRIBUTE_NODE	The attribute name	The attribute value
TEXT_NODE	#text	The text of the node
CDATA_SECTION_NODE	#cdata-section	The text of the node
ENTITY_REFERENCE_NODE	The name of the referenced entity	null
ENTITY_NODE	The entity name	null
PROCESSING_INSTRUCTION_NODE	The target of the PI	The remainder of the PI
COMMENT_NODE	#comment	The text of the comment
DOCUMENT_NODE	#document	null
DOCUMENT_TYPE_NODE	The document type name	null
DOCUMENT_FRAGMENT_NODE	#document-fragment	null
NOTATION_NODE	The notation name	null

See Also

Document, Element, Text; Chapter 17

Node.appendChild( )

DOM Level 1 Core

insert a node as the last child of this node

Synopsis

```
Node appendChild(Node newChild)
    throws DOMException;
```

Arguments

newChild

The node to be inserted into the document. If the node is a DocumentFragment, it is not directly inserted, but each of its children are.

Returns

The node that was added.

Throws

This method may throw a DOMException with one of the following code values in the following circumstances:

HIERARCHY_REQUEST_ERR
: The node does not allow children, or it does not allow children of the specified type, or *newChild* is an ancestor of this node (or is this node itself).

WRONG_DOCUMENT_ERR
: The ownerDocument property of *newChild* is not the same as the ownerDocument property of this node.

NO_MODIFICATION_ALLOWED_ERR
: This node is read-only and does not allow children to be appended, or the node being appended is already part of the document tree and its parent is read-only and does not allow children to be removed.

Description

This method adds the node *newChild* to the document, inserting it as the last child of this node. If *newChild* is already in the document tree, it is removed from the tree and then reinserted at its new location. If *newChild* is a DocumentFragment node, it is not inserted itself; instead, all its children are appended, in order, to the end of this node's childNodes[] array. Note that a node from (or created by) one document cannot be inserted into a different document. That is, the ownerDocument property of *newChild* must be the same as the ownerDocument property of this node.

Example

The following function inserts a new paragraph at the end of the document:

```
function appendMessage(message) {
    var pElement = document.createElement("P");
    var messageNode = document.createTextNode(message);
    pElement.appendChild(messageNode);    // Add text to paragraph
    document.body.appendChild(pElement);  // Add paragraph to document body
}
```

See Also

Node.insertBefore( ), Node.removeChild( ), Node.replaceChild( )

Node.cloneNode( )

DOM Level 1 Core

duplicate a node and, optionally, all of its descendants

Synopsis

```
Node cloneNode(boolean deep);
```

Arguments

deep
: If this argument is true, cloneNode( ) recursively clones all descendants of this node. Otherwise, it clones only this node.

Returns

A copy of this node.

Description

The `cloneNode( )` method makes and returns a copy of the node on which it is called. If passed the argument `true`, it recursively clones all descendants of the node as well. Otherwise, it clones only the node and none of its children. The returned node is not part of the document tree, and its `parentNode` property is `null`. When an Element node is cloned, all of its attributes are also cloned. Note, however, that EventListener functions registered on a node are not cloned.

Node.hasAttributes( ) DOM Level 2 Core

determine whether a node has attributes

Synopsis

```
boolean hasAttributes( );
```

Returns

`true` if this node has one or more attributes; `false` if it has none. Note that only Element nodes can have attributes.

See Also

Element.getAttribute( ), Element.hasAttribute( ), Node.attributes

Node.hasChildNodes( ) DOM Level 1 Core

determine whether a node has children

Synopsis

```
boolean hasChildNodes( );
```

Returns

`true` if this node has one or more children; `false` if it has none.

See Also

Node.childNodes

Node.insertBefore( ) DOM Level 1 Core

insert a node into the document tree before the specified node

Synopsis

```
Node insertBefore(Node newChild,
                  Node refChild)
    throws DOMException;
```

Arguments

newChild
: The node to be inserted into the tree. If it is a DocumentFragment, its children are inserted instead.

refChild
: The child of this node before which *newChild* is to be inserted. If this argument is `null`, *newChild* is inserted as the last child of this node.

Returns

The node that was inserted.

Throws

This method may throw a DOMException with the following `code` values:

`HIERARCHY_REQUEST_ERR`
: This node does not support children, or it does not allow children of the specified type, or *newChild* is an ancestor of this node (or is this node itself).

`WRONG_DOCUMENT_ERR`
: The `ownerDocument` property of *newChild* and this node are different.

`NO_MODIFICATION_ALLOWED_ERR`
: This node is read-only and does not allow insertions or the parent of *newChild* is read-only and does not allow deletions.

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

Description

This method inserts the node *newChild* into the document tree as a child of this node. The new node is positioned within this node's `childNodes[]` array so that it comes immediately before the *refChild* node. If *refChild* is `null`, *newChild* is inserted at the end of `childNodes[]`, just as with the `appendChild( )` method. Note that it is illegal to call this method with a *refChild* that is not a child of this node.

If *newChild* is already in the document tree, it is removed from the tree and then reinserted at its new position. If *newChild* is a DocumentFragment node, it is not inserted itself; instead, each of its children is inserted, in order, at the specified location.

Example

The following function inserts a new paragraph at the beginning of a document:

```
function insertMessage(message) {
    var paragraph = document.createElement("p");  // Create a <p> Element
    var text = document.createTextNode(message);  // Create a Text node
    paragraph.appendChild(text);                  // Add text to the paragraph
    // Now insert the paragraph before the first child of the body
    document.body.insertBefore(paragraph, document.body.firstChild)
}
```

See Also

Node.appendChild( ), Node.removeChild( ), Node.replaceChild( )

Node.isSupported( )

DOM Level 2 Core

determine if a node supports a feature

Synopsis

```
boolean isSupported(String feature,
                    String version);
```

Arguments

feature
: The name of the feature to test.

version
: The version number of the feature to test, or the empty string to test for support of any version of the feature.

Returns

`true` if the node supports the specified version of the specified feature, and `false` if it does not.

Description

The W3C DOM standard is modular, and implementations are not required to implement all modules or features of the standard. This method tests whether the implementation of this node supports the specified version of the named feature. See the "DOMImplementation.hasFeature( )" reference page for a list of values for the *feature* and *version* arguments.

See Also

DOMImplementation.hasFeature( )

Node.normalize( )

DOM Level 1 Core

merge adjacent Text nodes and remove empty ones

Synopsis

```
void normalize( );
```

Description

This method traverses all descendants of this node and "normalizes" the document by removing any empty Text nodes and merging all adjacent Text nodes into a single node. This can sometimes be useful to simplify the tree structure after node insertions or deletions.

See Also

Text

Node.removeChild( )

DOM Level 1 Core

remove (and return) the specified child of this node

Synopsis

```
Node removeChild(Node oldChild)
    throws DOMException;
```

Arguments

oldChild
: The child node to remove.

Returns

The node that was removed.

Throws

This method may throw a DOMException with the following code values in the following circumstances:

NO_MODIFICATION_ALLOWED_ERR
: This node is read-only and does not allow children to be removed.

NOT_FOUND_ERR
: *oldChild* is not a child of this node.

Description

This method removes the specified child from the childNodes[] array of this node. It is an error to call this method with a node that is not a child. removeChild( ) returns the *oldChild* node after removing it. *oldChild* continues to be a valid node and may be reinserted into the document later.

Example

You can delete the last child of the document body with this code:

```
document.body.removeChild(document.body.lastChild);
```

See Also

Node.appendChild( ), Node.insertBefore( ), Node.replaceChild( )

Node.replaceChild( )

DOM Level 1 Core

replace a child node with a new node

Synopsis

```
Node replaceChild(Node newChild,
                  Node oldChild)
    throws DOMException;
```

Arguments

newChild
: The replacement node.

oldChild
: The node to be replaced.

Returns

The node that was removed from the document and replaced.

Throws

This method may throw a DOMException with the following code values:

HIERARCHY_REQUEST_ERR
: This node does not allow children, or does not allow children of the specified type, or *newChild* is an ancestor of this node (or is this node itself).

WRONG_DOCUMENT_ERR
: *newChild* and this node have different values for ownerDocument.

NO_MODIFICATION_ALLOWED_ERR
: This node is read-only and does not allow replacement, or *newChild* is the child of a node that does not allow removals.

NOT_FOUND_ERR
: *oldChild* is not a child of this node.

Description

This method replaces one node of the document tree with another. *oldChild* is the node to be replaced, and must be a child of this node. *newChild* is the node that takes its place in the childNodes[] array of this node.

If *newChild* is already part of the document, it is first removed from the document before being reinserted at its new position. If *newChild* is a DocumentFragment, it is not inserted itself; instead each of its children is inserted, in order, at the position formerly occupied by *oldChild*.

Example

The following code replaces a node n with a <b> element and then inserts the replaced node into the <b> element, which reparents the node and makes it appear in bold:

```
// Get the first child node of the first paragraph in the document
var n = document.getElementsByTagName("p")[0].firstChild;
var b = document.createElement("b");  // Create a <b> element
n.parentNode.replaceChild(b, n);      // Replace the node with <b>
b.appendChild(n);                     // Reinsert the node as a child of <b>
```

See Also

Node.appendChild( ), Node.insertBefore( ), Node.removeChild( )

NodeFilter

DOM Level 2 Traversal

a function to filter the nodes of a document tree

Constants

The following three constants are the legal return values for node filter functions. Note that they are static properties of the object named NodeFilter, not properties of individual node filter functions:

`short FILTER_ACCEPT = 1`
: Accept this node. A NodeIterator or TreeWalker will return this node as part of its document traversal.

`short FILTER_REJECT = 2`
: Reject this node. A NodeIterator or TreeWalker will behave as if this node does not exist. Furthermore, this return value tells a TreeWalker to ignore all children of this node.

`short FILTER_SKIP = 3`
: Skip this node. A NodeIterator or TreeWalker will not return this node, but it will recursively consider its children as part of document traversal.

The following constants are bit flags that can be set in the *whatToShow* argument to the `createNodeIterator()` and `createTreeWalker()` methods of the Document object. Each constant corresponds to one of the types of Document nodes (see the "Node" reference page for a list of node types) and specifies that a NodeIterator or TreeWalker should consider nodes of that type during its traversal of the document. Multiple constants can be combined using the logical OR operator `|`. `SHOW_ALL` is a special value with all bits set: it indicates that all nodes should be considered, regardless of their type.

```
unsigned long SHOW_ALL = 0xFFFFFFFF;
unsigned long SHOW_ELEMENT = 0x00000001;
unsigned long SHOW_ATTRIBUTE = 0x00000002;
unsigned long SHOW_TEXT = 0x00000004;
unsigned long SHOW_CDATA_SECTION = 0x00000008;
unsigned long SHOW_ENTITY_REFERENCE = 0x00000010;
unsigned long SHOW_ENTITY = 0x00000020;
unsigned long SHOW_PROCESSING_INSTRUCTION = 0x00000040;
unsigned long SHOW_COMMENT = 0x00000080;
unsigned long SHOW_DOCUMENT = 0x00000100;
unsigned long SHOW_DOCUMENT_TYPE = 0x00000200;
unsigned long SHOW_DOCUMENT_FRAGMENT = 0x00000400;
unsigned long SHOW_NOTATION = 0x00000800;
```

Methods

`acceptNode()`
: In languages such as Java that do not allow functions to be passed as arguments, you define a node filter by defining a class that implements this interface and includes an implementation for this function. The function is passed a node and must return one of the constants `FILTER_ACCEPT`, `FILTER_REJECT`, or `FILTER_SKIP`. In JavaScript, however,

you create a node filter simply by defining a function (with any name) that accepts a node argument and returns one of the three filter constants. See the following sections for details and an example.

Description

A node filter is an object that can examine a Document node and tell a NodeIterator or TreeWalker whether to include the node in its document traversal. In JavaScript, a node filter is simply a function that takes a single node argument and returns one of the three `FILTER_` constants defined earlier. There is no NodeFilter interface; there is simply an object named NodeFilter that has properties that define those constants. To use a node filter, you pass it to the `createNodeIterator()` or `createTreeWalker()` method of the Document object. Your node filter function will then be called to evaluate nodes when you use the resulting NodeIterator or TreeWalker object to traverse the document.

Node filter functions should ideally be written so that they do not themselves alter the document tree and do not throw any exceptions. Also, node filters are not allowed to base their filtering decisions on the history of past invocations of those filters.

Example

You might define and use a node filter function as follows:

```
// Define a node filter that filters out everything but <h1> and <h2> elements
var myfilter = function(n) {  // Filter node n
    if ((n.nodeName == "H1") || (n.nodeName == "H2"))
        return NodeFilter.FILTER_ACCEPT;
    else
        return NodeFilter.FILTER_SKIP;
}

// Now create a NodeIterator that uses the filter
var ni = document.createNodeIterator(document.body,  // Traverse the document body
                                     NodeFilter.SHOW_ELEMENT,  // Elements only
                                     myfilter,  // Filter by tag name
                                     false);  // No entity expansion
```

See Also

NodeIterator, TreeWalker

Type of: NodeIterator.filter, TreeWalker.filter

Passed to: Document.createNodeIterator(), Document.createTreeWalker()

NodeIterator

DOM Level 2 Traversal

iterate through a filtered sequence of Document nodes

Properties

`readonly boolean expandEntityReferences`

Whether this NodeIterator traverses the children of EntityReference nodes (in XML documents). The value is specified as an argument to `Document.createNodeIterator()` when the NodeIterator is first created.

`readonly NodeFilter filter`
: The node filter function that was specified for this NodeIterator in the call to `Document.createNodeIterator( )`.

`readonly Node root`
: The root node at which the NodeIterator begins iterating. The value of this property is specified in the call to `Document.createNodeIterator( )`.

`readonly unsigned long whatToShow`
: A set of bit flags (see "NodeFilter" for a list of valid flags) that specifies what types of Document nodes this NodeIterator will consider. If a bit is not set in this property, the corresponding node type will always be ignored by this NodeIterator. Note that the value of this property is specified in the call to `Document.createNodeIterator( )`.

Methods

`detach( )`
: "Detaches" this NodeIterator from its document so that the implementation no longer needs to modify the NodeIterator when the document is modified. Call this method when you are done using a NodeIterator. After `detach( )` has been called, any calls to other NodeIterator methods will cause exceptions.

`nextNode( )`
: Returns the next node in the filtered sequence of nodes represented by this NodeIterator, or `null` if the NodeIterator has already returned the last node.

`previousNode( )`
: Returns the previous node in the filtered sequence of nodes represented by this NodeIterator, or `null` if there is no previous node.

Description

A NodeIterator represents the sequence of Document nodes that results from traversing a document subtree in document source order and filtering the nodes using a two-stage process. Create a NodeIterator object with `Document.createNodeIterator( )`. Use the `nextNode( )` and `previousNode( )` methods to iterate forward and backward through the sequence of nodes. Call `detach( )` when you are done with a NodeIterator, unless you are sure that the NodeIterator will be garbage collected before the document is modified. Note that the properties of this interface are all read-only copies of the arguments passed to `Document.createNodeIterator( )`.

To be returned by the `nextNode( )` or `previousNode( )` methods, a node must pass two filtration steps. First, the node type must be one of the types specified by the `whatToShow` property. See "NodeFilter" for a list of constants that can be combined to specify the *`whatToShow`* argument to `Document.createNodeIterator( )`. Next, if the `filter` property is not `null`, each node that passes the `whatToShow` test is passed to the filter function specified by the `filter` property. If this function returns `NodeFilter.FILTER_ACCEPT`, the node is returned. If it returns `NodeFilter.FILTER_REJECT` or `NodeFilter.FILTER_SKIP`, the NodeIterator skips the node. Note that when a node is rejected by either of these filtration steps, it is only the node itself that is rejected; the children of the node are not automatically rejected and are subject to the same filtration steps.

NodeIterator objects remain valid even if the document tree they are traversing is modified. The `nextNode( )` and `previousNode( )` methods return nodes based on the current

state of the document, not the state of the document that existed when the NodeIterator was created.

See Also

NodeFilter, TreeWalker; Chapter 17

Returned by: Document.createNodeIterator( )

NodeIterator.detach( )

DOM Level 2 Traversal

free a NodeIterator object

Synopsis

```
void detach( );
```

Description

DOM implementations keep track of all NodeIterator objects created for a document, because they may need to modify the state of the NodeIterator when certain Document nodes are deleted. When you are certain that a NodeIterator isn't needed anymore, call `detach( )` to tell the implementation that it no longer needs to keep track of it. Note, however, that once you call this method any subsequent call to `nextNode( )` or `previousNode( )` will throw an exception.

Calling `detach( )` is not required, but doing so may improve performance when the document is being modified and the NodeIterator object is not immediately garbage collected.

NodeIterator.nextNode( )

DOM Level 2 Traversal

iterate to the next node

Synopsis

```
Node nextNode( )
    throws DOMException;
```

Returns

The next node in the sequence of nodes represented by this NodeIterator, or `null` if the last node has already been returned.

Throws

If this method is called after a call to `detach( )`, it throws a DOMException with a `code` of `INVALID_STATE_ERR`.

Description

This method iterates forward through the sequence of nodes represented by this NodeIterator. If this is the first time it is called for a NodeIterator, it returns the first node in the sequence. Otherwise, it returns the node that follows the one that was previously returned.

Example

```
// Create a NodeIterator to represent all elements in the document body
var ni = document.createNodeIterator(document.body, NodeFilter.SHOW_ELEMENT,
                                     null, false);
// Loop forward through all nodes in the iterator
for(var e = ni.nextNode(); e != null; e = ni.nextNode()) {
    // Do something with element e
}
```

NodeIterator.previousNode( )

DOM Level 2 Traversal

iterate to the previous node

Synopsis

```
Node previousNode( )
    throws DOMException;
```

Returns

The previous node in the sequence of nodes represented by this NodeIterator, or `null` if there is no previous node.

Throws

If this method is called after a call to `detach( )`, it throws a DOMException with a `code` of `INVALID_STATE_ERR`.

Description

This method iterates backward through the sequence of nodes represented by this NodeIterator. It returns the node before the one that was most recently returned by `previousNode( )` or `nextNode( )`. If there is no such node in the sequence, it returns `null`.

NodeList

DOM Level 1 Core

a read-only array of nodes

Properties

`readonly unsigned long length`
: The number of nodes in the array.

Methods

`item( )`
: Returns the specified element of the array.

Description

The NodeList interface defines a read-only ordered list (i.e., an array) of Node objects. The `length` property specifies how many nodes are in the list, and the `item( )` method allows

you to obtain the node at a specified position in the list. The elements of a NodeList are always valid Node objects: NodeLists never contain `null` elements.

In JavaScript, NodeList objects behave like JavaScript arrays, and you can query an element from the list using square-bracket array notation instead of calling the `item( )` method. However, you cannot assign new nodes to a NodeList using square brackets. Since it is always easier to think of a NodeList object as a read-only JavaScript array, this book uses the notation `Node[]` (i.e., a Node array) instead of NodeList. See "Element. getElementsByTagName( )," for example, which is listed as returning a `Node[]` instead of a NodeList object. Similarly, the `childNodes` property of the Node object is technically a NodeList object, but the "Node" reference page defines it as a `Node[]`, and the property itself is usually referred to as "the `childNodes[]` array."

Note that NodeList objects are "live": they are not static, but immediately reflect changes to the document tree. For example, if you have a NodeList that represents the children of a specific node and you then delete one of those children, the child will be removed from your NodeList. Be careful when you are looping through the elements of a NodeList if the body of your loop makes changes to the document tree (such as deleting nodes) that may affect the contents of the NodeList!

See Also

NamedNodeMap

Type of: Node.childNodes

Returned by: Document.getElementsByTagName( ), Document.getElementsByTagNameNS( ), Element.getElementsByTagName( ), Element.getElementsByTagNameNS( ), HTMLDocument.getElementsByName( )

NodeList.item( )

DOM Level 1 Core

get an element of a NodeList

Synopsis

```
Node item(unsigned long index);
```

Arguments

index

The position (or index) of the desired node in the NodeList. The index of the first node in the NodeList is 0, and the index of the last Node is `length`–1.

Returns

The node at the specified position in the NodeList, or `null` if *index* is less than zero or greater than or equal to the length of the NodeList.

Description

This method returns the specified element of a NodeList. In JavaScript, you can use the square-bracket array notation instead of calling `item( )`.

Notation

DOM Level 1 XML

a notation in an XML DTD

Node → Notation

Properties

`readonly String publicId`
: The public identifier of the notation, or `null` if none is specified.

`readonly String systemId`
: The system identifier of the notation, or `null` if none is specified.

Description

This infrequently used interface represents a notation declaration in the document type definition (DTD) of an XML document. In XML, notations are used to specify the format of an unparsed entity or to formally declare a processing instruction target.

The name of the notation is specified by the inherited `nodeName` property. Because notations appear in the DTD and not the document itself, Notation nodes are never part of the document tree, and the `parentNode` property is always null. The `notations` property of the DocumentType interface provides a way to look up Notation objects by notation name.

Notation objects are read-only and cannot be modified in any way.

See Also

DocumentType

ProcessingInstruction

DOM Level 1 XML

a processing instruction in an XML document

Node → ProcessingInstruction

Properties

`String data`
: The content of the processing instruction (i.e., the first non-space character after the target up to but not including the closing ?>).

`readonly String target`
: The target of the processing instruction. This is the first identifier that follows the opening <?; it specifies the "processor" for which the processing instruction is intended.

Description

This infrequently used interface represents a processing instruction (or PI) in an XML document. Programmers working with HTML documents will never encounter a ProcessingInstruction node.

See Also

Returned by: Document.createProcessingInstruction()

Range

DOM Level 2 Range

represents a contiguous range of a document

Constants

These constants specify how the boundary points of two Range objects are to be compared. They are the legal values for the *how* argument to the `compareBoundaryPoints( )` method. See the "Range.compareBoundaryPoints()" reference page.

`unsigned short START_TO_START = 0`
: Compare the start of the specified range to the start of this range.

`unsigned short START_TO_END = 1`
: Compare the start of the specified range to the end of this range.

`unsigned short END_TO_END = 2`
: Compare the end of the specified range to the end of this range.

`unsigned short END_TO_START = 3`
: Compare the end of the specified range to the start of this range.

Properties

The Range interface defines the following properties. Note that all of these properties are read-only. You cannot change the start or end points of a range by setting properties; you must call `setEnd( )` or `setStart( )` instead. Note also that after you call the `detach( )` method of a Range object, any subsequent attempt to read any of these properties throws a DOMException with a `code` of `INVALID_STATE_ERR`.

`readonly boolean collapsed`
: `true` if the start and the end of the range are at the same point in the document—that is, if the range is empty or "collapsed."

`readonly Node commonAncestorContainer`
: The most deeply nested Document node that contains (i.e., is an ancestor of) both the start and end points of the range.

`readonly Node endContainer`
: The Document node that contains the end point of the range.

`readonly long endOffset`
: The end point position within `endContainer`.

`readonly Node startContainer`
: The Document node that contains the starting point of the range.

`readonly long startOffset`
: The position of the range's starting point within `startContainer`.

Methods

The Range interface defines the following methods. Note that if you call `detach( )` on a range, any subsequent calls of any methods on that range throw a DOMException with a `code` of `INVALID_STATE_ERR`. Because this exception is ubiquitous within this interface, it is not listed in the reference pages for the individual Range methods.

`cloneContents( )`
: Returns a new DocumentFragment object that contains a copy of the region of the document represented by this range.

`cloneRange( )`
: Creates a new Range object that represents the same region of the document as this one.

`collapse( )`
: Collapses this range so that one boundary point is the same as the other.

`compareBoundaryPoints( )`
: Compares a boundary point of the specified range to a boundary point of this range, and returns –1, 0, or 1, depending on their order. Which points to compare is specified by the first argument, which must be one of the previously defined constants.

`deleteContents( )`
: Deletes the region of the document represented by this range.

`detach( )`
: Tells the implementation that this range will no longer be used and that it can stop keeping track of it. If you call this method for a range, subsequent method calls or property lookups on that range throw a DOMException with a `code` of `INVALID_STATE_ERR`.

`extractContents( )`
: Deletes the region of the document represented by this range, but returns the contents of that region as a DocumentFragment object. This method is like a combination of `cloneContents( )` and `deleteContents( )`.

`insertNode( )`
: Inserts the specified node into the document at the start point of the range.

`selectNode( )`
: Sets the boundary points of this range so that it contains the specified node and all of its descendants.

`selectNodeContents( )`
: Sets the boundary points of this range so that it contains all the descendants of the specified node but not the node itself.

`setEnd( )`
: Sets the end point of this range to the specified node and offset.

`setEndAfter( )`
: Sets the end point of this range to immediately after the specified node.

`setEndBefore( )`
: Sets the end point of this range to immediately before the specified node.

`setStart( )`
: Sets the start position of this range to the specified offset within the specified node.

`setStartAfter( )`
: Sets the start position of this range to immediately after the specified node.

`setStartBefore( )`
: Sets the start position of this range to immediately before the specified node.

`surroundContents( )`
: Inserts the specified node into the document at the start position of the range and then reparents all the nodes within the range so that they become descendants of the newly inserted node.

`toString( )`
: Returns the plain-text content of the document region described by this range.

Description

A Range object represents a contiguous range or region of a document, such as the region that the user might select with a mouse drag in a web browser window. If an implementation supports the Range module, the Document object defines a `createRange( )` method that you can call to create a new Range object. (Be careful, however: Internet Explorer defines an incompatible `Document.createRange( )` method that returns an object similar to, but not compatible with, the Range interface.) The Range interface defines a number of methods for specifying a "selected" region of a document and several more methods for implementing cut and paste–type operations on the selected region.

A range has two boundary points: a start point and an end point. Each boundary point is specified by a combination of a node and an offset within that node. The node is typically an Element, Document, or Text node. For Element and Document nodes, the offset refers to the children of that node. An offset of 0 specifies a boundary point before the first child of the node. An offset of 1 specifies a boundary point after the first child and before the second child. If the boundary node is a Text node, however, the offset specifies a position between two characters of that text.

The properties of the Range interface provide a way to obtain boundary nodes and offsets of a range. The methods of the interface provide a number of ways to set the boundaries of a range. Note that the boundaries of a range may be set to nodes within a Document or a DocumentFragment.

Once the boundary points of a range are defined, you can use `deleteContents( )`, `extractContents( )`, `cloneContents( )`, and `insertNode( )` to implement cut-, copy-, and paste-style operations.

When a document is altered by insertion or deletion, all Range objects that represent portions of that document are altered, if necessary, so that their boundary points remain valid and they represent (as closely as possible) the same document content.

For further details, read the reference pages for each of the Range methods and see the discussion of the Range API in Chapter 17.

See Also

Document.createRange( ), DocumentFragment; Chapter 17

Passed to: Range.compareBoundaryPoints( )

Returned by: Document.createRange( ), Range.cloneRange( )

Range.cloneContents( )

DOM Level 2 Range

copy range contents into a DocumentFragment

Synopsis

```
DocumentFragment cloneContents( )
    throws DOMException;
```

Returns

A DocumentFragment object that contains a copy of the document content within this range.

Throws

If this range includes a DocumentType node, this method throws a DOMException with a code of `HIERARCHY_REQUEST_ERR`.

Description

This method duplicates the contents of this range and returns the results in a DocumentFragment object.

See Also

DocumentFragment, Range.deleteContents( ), Range.extractContents( )

Range.cloneRange( )

DOM Level 2 Range

make a copy of this range

Synopsis

```
Range cloneRange( );
```

Returns

A new Range object that has the same boundary points as this range.

See Also

Document.createRange( )

Range.collapse( )

DOM Level 2 Range

make one boundary point equal to the other

Synopsis

```
void collapse(boolean toStart)
    throws DOMException;
```

Arguments

toStart
: If this argument is `true`, the method sets the end point of the range to the same value as the starting point. Otherwise, it sets the starting point to the same value as the end point.

Description

This method sets one boundary point of the range to be the same as the other point. The point to be modified is specified by the *toStart* argument. After this method returns, the range is said to be "collapsed": it represents a single point within a document and has no content. When a range is collapsed like this, its `collapsed` property is `true`.

Range.compareBoundaryPoints( ) DOM Level 2 Range

compare positions of two ranges

Synopsis

```
short compareBoundaryPoints(unsigned short how,
                            Range sourceRange)
    throws DOMException;
```

Arguments

how
: Specifies how to perform the comparison (i.e., which boundary points to compare). Legal values are the constants defined by the Range interface.

sourceRange
: The range that is to be compared to this range.

Returns

–1 if the specified boundary point of this range is before the specified boundary point of `sourceRange`, 0 if the two specified boundary points are the same, or 1 if the specified boundary point of this range is after the specified boundary point of `sourceRange`.

Throws

If `sourceRange` represents a range of a different document than this range does, this method throws a DOMException with a `code` of `WRONG_DOCUMENT_ERR`.

Description

This method compares a boundary point of this range to a boundary point of the specified `sourceRange` and returns a value that specifies their relative order in the document source. The *how* argument specifies which boundary points of each range are to be compared. The legal values for this argument, and their meanings, are as follows:

`Range.START_TO_START`
: Compare the start points of the two Range nodes.

`Range.END_TO_END`
: Compare the end points of the two Range nodes.

`Range.START_TO_END`
: Compare the start point of *sourceRange* to the end point of this range.

`Range.END_TO_START`
: Compare the end point of *sourceRange* to the start point of this range.

The return value of this method is a number that specifies the relative position of this range to the specified *sourceRange*. Therefore, you might expect the range constants for the *how* argument to specify the boundary point for this range first and the boundary point for `sourceRange` second. Counterintuitively, however, the `Range.START_TO_END` constant specifies a comparison of the *end* point of this range with the *start* point of the specified *sourceRange*. Similarly, the `Range.END_TO_START` constant specifies a comparison of the *start* point of this range with the *end* point of the specified range.

Range.deleteContents( )

DOM Level 2 Range

delete a region of the document

Synopsis

```
void deleteContents( )
    throws DOMException;
```

Throws

If any portion of the document that is represented by this range is read-only, this method throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR`.

Description

This method deletes all document content represented by this range. When this method returns, the range is collapsed with both boundary points at the same position. Note that the deletion may result in adjacent Text nodes that can be merged with a call to `Node.normalize( )`.

See "cloneContents( )" for a way to copy document content and "extractContents( )" for a way to copy and delete document content in a single operation.

See Also

Node.normalize( ), Range.cloneContents( ), Range.extractContents( )

Range.detach( )

DOM Level 2 Range

free a Range object

Synopsis

```
void detach( )
    throws DOMException;
```

Throws

Like all Range methods, `detach( )` throws a DOMException with a `code` of `INVALID_STATE_ERR` if it is called on a Range object that has already been detached.

Description

DOM implementations keep track of all Range objects created for a document, because they may need to change the range boundary points when the document is modified. When you are certain that a Range object isn't needed any more, call the `detach( )` method to tell the implementation that it no longer needs to keep track of that range. Note that once this method has been called for a Range object, any use of that Range will throw an exception. Calling `detach( )` is not required but may improve performance in some circumstances when the document is being modified and a Range object is not subject to immediate garbage collection.

Range.extractContents( )

DOM Level 2 Range

delete document content and return it in a DocumentFragment

Synopsis

```
DocumentFragment extractContents( )
    throws DOMException;
```

Returns

A DocumentFragment node that contains the contents of this range.

Throws

This method throws a DOMException with a `code` of `NO_MODIFICATION_ALLOWED_ERR` if any part of the document content to be extracted is read-only, or a `code` of `HIERARCHY_REQUEST_ERR` if the range contains a DocumentType node.

Description

This method deletes the specified range of a document and returns a DocumentFragment node that contains the deleted content (or a copy of the deleted content). When this method returns, the range is collapsed, and the document may contain adjacent Text nodes (which can be merged with `Node.normalize( )`).

See Also

DocumentFragment, Range.cloneContents( ), Range.deleteContents( )

Range.insertNode( )

DOM Level 2 Range

insert a node at the start of a range

Synopsis

```
void insertNode(Node newNode)
    throws RangeException,
           DOMException;
```

Arguments

newNode
: The node to be inserted into the document.

Throws

This method throws a RangeException with a code of INVALID_NODE_TYPE_ERR if newNode is an Attr, Document, Entity, or Notation node.

This method also throws a DOMException with one of the following code values under the following conditions:

HIERARCHY_REQUEST_ERR
: The node that contains the start of the range does not allow children, or it does not allow children of the specified type, or *newNode* is an ancestor of that node.

NO_MODIFICATION_ALLOWED_ERR
: The node that contains the start of the range, or any of its ancestors, is read-only.

WRONG_DOCUMENT_ERR
: newNode is part of a different document than the range is.

Description

This method inserts the specified node (and all its descendants) into the document at the start position of this range. When this method returns, this range includes the newly inserted node. If *newNode* is already part of the document, it is removed from its current position and then reinserted at the start of the range. If *newNode* is a DocumentFragment node, it is not inserted itself, but all of its children are inserted, in order, at the start of the range.

If the node that contains the start of the range is a Text node, it is split into two adjacent nodes before the insertion takes place. If *newNode* is a Text node, it is not merged with any adjacent Text nodes after it is inserted. To merge adjacent nodes, call Node.normalize( ).

See Also

DocumentFragment, Node.normalize( )

Range.selectNode( )

DOM Level 2 Range

set range boundaries to a node

Synopsis

```
void selectNode(Node refNode)
    throws RangeException,
           DOMException;
```

Arguments

refNode
: The node to be "selected" (i.e., the node that is to become the content of this range).

Throws

A RangeException with a code of INVALID_NODE_TYPE_ERR if *refNode* is an Attr, Document, DocumentFragment, Entity, or Notation node, or if any ancestor of *refNode* is a DocumentType, Entity, or Notation node.

A DOMException with a code of WRONG_DOCUMENT_ERR if *refNode* is part of a different document than the one through which this range was created.

Description

This method sets the contents of this range to the specified *refNode*. That is, it "selects" the node and its descendants.

See Also

Range.selectNodeContents( )

Range.selectNodeContents( )

DOM Level 2 Range

set range boundaries to the children of a node

Synopsis

```
void selectNodeContents(Node refNode)
    throws RangeException,
           DOMException;
```

Arguments

refNode
The node whose children are to become the contents of this range.

Throws

A RangeException with a code of `INVALID_NODE_TYPE_ERR` if *refNode* or one of its ancestors is a DocumentType, Entity, or Notation node.

A DOMException with a code of `WRONG_DOCUMENT_ERR` if *refNode* is part of a different document than the one through which this range was created.

Description

This method sets the boundary points of this range so that the range contains the children of *refNode*.

See Also

Range.selectNode( )

Range.setEnd( )

DOM Level 2 Range

set the end point of a range

Synopsis

```
void setEnd(Node refNode,
            long offset)
    throws RangeException,
           DOMException;
```

Arguments

refNode
The node that contains the new end point.

offset
The position of the end point within *refNode*.

Throws

A RangeException with a code of INVALID_NODE_TYPE_ERR if *refNode* or one of its ancestors is a DocumentType, Entity, or Notation node.

A DOMException with a code of WRONG_DOCUMENT_ERR if *refNode* is part of a different document than the one through which this range was created, or a code of INDEX_SIZE_ERR if *offset* is negative or is greater than the number of children or characters in *refNode*.

Description

This method sets the end point of a range by specifying the values of the endContainer and endOffset properties.

Range.setEndAfter( )

DOM Level 2 Range

end a range after a specified node

Synopsis

```
void setEndAfter(Node refNode)
    throws RangeException,
           DOMException;
```

Arguments

refNode
The node after which the end point of the range is to be set.

Throws

A RangeException with a code of INVALID_NODE_TYPE_ERR if *refNode* is a Document, DocumentFragment, Attr, Entity, or Notation node, or if the root container of *refNode* is not a Document, DocumentFragment, or Attr node.

A DOMException with a code of WRONG_DOCUMENT_ERR if *refNode* is part of a different document than the one through which this range was created.

Description

This method sets the end point of this range to fall immediately after the specified *refNode*.

Range.setEndBefore( )

DOM Level 2 Range

end a range before the specified node

Synopsis

```
void setEndBefore(Node refNode)
    throws RangeException,
           DOMException;
```

Arguments

refNode
: The node before which the end point of the range is to be set.

Throws

This method throws the same exceptions in the same circumstances as `Range.setEndAfter( )`. See that method for details.

Description

This method sets the end point of this range to fall immediately before the specified *refNode*.

Range.setStart( )

DOM Level 2 Range

set the start point of a range

Synopsis

```
void setStart(Node refNode,
              long offset)
    throws RangeException,
           DOMException;
```

Arguments

refNode
: The node that contains the new start point.

offset
: The position of the new start point within *refNode*.

Throws

This method throws the same exceptions, for the same reasons, as `Range.setEnd( )`. See that method for details.

Description

This method sets the start point of this range by specifying the values of the `startContainer` and `startOffset` properties.

Range.setStartAfter( )

DOM Level 2 Range

start a range after the specified node

Synopsis

```
void setStartAfter(Node refNode)
    throws RangeException,
           DOMException;
```

Arguments

refNode
: The node after which the start point of the range is to be set.

Throws

This method throws the same exceptions in the same circumstances as `Range.setEndAfter( )`. See that method for details.

Description

This method sets the starting point of this range to be immediately after the specified *refNode*.

Range.setStartBefore( )

DOM Level 2 Range

start a range before the specified node

Synopsis

```
void setStartBefore(Node refNode)
    throws RangeException,
           DOMException;
```

Arguments

refNode
: The node before which the start point of the range is to be set.

Throws

This method throws the same exceptions in the same circumstances as `Range.setEndAfter( )`. See that method for details.

Description

This method sets the starting point of this range to be immediately before the specified *refNode*.

Range.surroundContents( )

DOM Level 2 Range

surround range contents with the specified node

Synopsis

```
void surroundContents(Node newParent)
    throws RangeException,
           DOMException;
```

Arguments

newParent
: The node that is to become the new parent of the contents of this range.

Throws

This method throws a DOMException or RangeException with one of the following code values in the following circumstances:

`DOMException.HIERARCHY_REQUEST_ERR`
: The container node of the start of the range does not allow children or does not allow children of the type of *newParent*, or *newParent* is an ancestor of that container node.

`DOMException.NO_MODIFICATION_ALLOWED_ERR`
: An ancestor of a boundary point of the range is read-only and does not allow insertions.

`DOMException.WRONG_DOCUMENT_ERR`
: *newParent* and this range were created using different Document objects.

`RangeException.BAD_BOUNDARYPOINTS_ERR`
: The range partially selects a node (other than a Text node), so the region of the document it represents cannot be surrounded.

`RangeException.INVALID_NODE_TYPE_ERR`
: *newParent* is a Document, DocumentFragment, DocumentType, Attr, Entity, or Notation node.

Description

This method reparents the contents of this range to *newParent* and then inserts *newParent* into the document at the start position of the range. It is useful to place a region of document content within a <b> or element, for example.

If *newParent* is already part of the document, it is first removed from the document and any children it has are discarded. When this method returns, this range begins immediately before newParent and ends immediately after it.

Range.toString( )

DOM Level 2 Range

get range contents as a plain-text string

Synopsis

```
String toString( );
```

Returns

The contents of this range as a string of plain text without any markup.

RangeException

DOM Level 2 Range

signals a range-specific exception

Constants

The following constants define the legal values for the code property of a RangeException object. Note that these constants are static properties of RangeException, not properties of individual exception objects.

`unsigned short BAD_BOUNDARYPOINTS_ERR = 1`
: The boundary points of a range are not legal for the requested operation.

`unsigned short INVALID_NODE_TYPE_ERR = 2`
: An attempt was made to set the container node of a range boundary point to an invalid node or a node with an invalid ancestor.

Properties

`unsigned short code`
: An error code that provides some detail about what caused the exception. The legal values (and their meanings) for this property are defined by the constants just listed.

Description

A RangeException is thrown by certain methods of the Range interface to signal a problem of some sort. Note that most exceptions thrown by Range methods are DOMException objects. A RangeException is generated only when none of the existing DOMException error constants is appropriate to describe the exception.

Rect

DOM Level 2 CSS

a CSS rect( ) value

Properties

`readonly CSSPrimitiveValue bottom`
: The bottom of the rectangle.

`readonly CSSPrimitiveValue left`
: The left side of the rectangle.

`readonly CSSPrimitiveValue right`
: The right side of the rectangle.

`readonly CSSPrimitiveValue top`
: The top of the rectangle.

Description

This interface represents a CSS `rect(top right bottom left)` value, as used with the CSS `clip` attribute, for example. Consult a CSS reference for details.

See Also

Returned by: CSSPrimitiveValue.getRectValue( )

RGBColor

DOM Level 2 CSS

a CSS color value

Properties

`readonly CSSPrimitiveValue blue`
: The blue component of the color.

`readonly CSSPrimitiveValue green`
The green component of the color.

`readonly CSSPrimitiveValue red`
The red component of the color.

Description

This interface represents a color specified in the RGB color space. The properties are CSSPrimitiveValue objects that specify the values of the red, green, and blue components of the color. Each CSSPrimitiveValue holds a number in the range 0–255 or a percentage in the range 0–100%.

See Also

Returned by: CSSPrimitiveValue.getRGBColorValue( )

StyleSheet

DOM Level 2 StyleSheets

a style sheet of any type

Subinterfaces

CSSStyleSheet

Properties

`boolean disabled`
If `true`, the style sheet is disabled and is not applied to the document. If `false`, the style sheet is enabled and is applied to the document (unless the `media` property specifies that the style sheet should not be applied to documents of this type).

`readonly String href`
The URL of a style sheet that is linked into the document, or `null` for inline style sheets.

`readonly MediaList media`
A list of media types for which this style sheet should be applied. If no media information is supplied for the style sheet, this property is a valid MediaList object that has a `length` of 0.

`readonly Node ownerNode`
The Document node that links the style sheet into the document, or the node that contains an inline style sheet. In HTML documents, this property refers to a `<link>` or `<style>` element. For style sheets that are included within other style sheets rather than directly in the document, this property is `null`.

`readonly StyleSheet parentStyleSheet`
The style sheet that included this one, or `null` if this style sheet was included directly in the document. See also "CSSStyleSheet.ownerRule."

`readonly String title`
The title of the style sheet, if one is specified. A title may be specified by the `title` attribute of a `<style>` or `<link>` tag that is the `ownerNode` of this style sheet.

`readonly String type`
The type of this style sheet, as a MIME type. CSS style sheets have a type of "text/css".

Description

This interface represents a style sheet that is associated with this document. If this is a CSS style sheet, the object that implements this StyleSheet interface also implements the CSSStyleSheet subinterface and defines properties and methods that allow you to examine and set the CSS rules that comprise the style sheet. See "CSSStyleSheet" for details.

If a DOM implementation supports style sheets, you can obtain a complete list of the style sheets associated with a document through the `Document.styleSheets` property. Furthermore, the HTML `<style>` and `<link>` elements and XML style-sheet ProcessingInstruction nodes implement the `LinkStyle` interface and provide a reference to the StyleSheet through a property named `sheet`.

See Also

CSSStyleSheet, Document.styleSheets, LinkStyle

Type of: LinkStyle.sheet, StyleSheet.parentStyleSheet

Returned by: StyleSheetList.item( )

StyleSheetList

DOM Level 2 StyleSheets

an array of style sheets

Properties

`readonly unsigned long length`
: The number of StyleSheet objects in the array.

Methods

`item( )`
: Returns the StyleSheet object at the specified position in the array, or `null` if the specified position is negative or is greater than or equal to `length`.

Description

This interface defines an array of StyleSheet objects. `length` specifies the number of style sheets in the array, and `item( )` provides a way to retrieve the style sheet at a given position. In JavaScript, you can treat a StyleSheetList object as a read-only array, and you can index it using ordinary square-bracket array notation instead of calling the `item( )` method.

See Also

Type of: DocumentStyle.styleSheets

StyleSheetList.item( )

DOM Level 2 StyleSheets

index an array of style sheets

Synopsis

```
StyleSheet item(unsigned long index);
```

Arguments

index
: The position of the desired style sheet within the array.

Returns

The StyleSheet object at the specified position within the array, or `null` if *index* is negative or is greater than or equal to `length`. Note that in JavaScript, it is usually simpler to treat a StyleSheetList object as an array and index it using square-bracket array notation instead of calling this method.

Text

DOM Level 1 Core

a run of text in an HTML or XML document

Node → CharacterData → Text

Subinterfaces

CDATASection

Methods

`splitText( )`
: Splits this Text node into two at the specified character position and returns the new Text node.

Description

A Text node represents a run of plain text in an HTML or XML document. Plain text appears within HTML and XML elements and attributes, and Text nodes typically appear as children of Element and Attr nodes. Text nodes inherit from CharacterData, and the textual content of a Text node is available through the `data` property inherited from CharacterData or through the `nodeValue` property inherited from Node. Text nodes may be manipulated using any of the methods inherited from CharacterData or with the `splitText( )` method defined by the Text interface itself. Text nodes never have children.

See "Node.normalize( )" for a way to remove empty Text nodes and merge adjacent Text nodes from a subtree of a document.

See Also

CharacterData, Node.normalize( )

Returned by: Document.createTextNode( ), Text.splitText( )

Text.splitText( )

DOM Level 1 Core

split a Text node in two

Synopsis

```
Text splitText(unsigned long offset)
    throws DOMException;
```

Arguments

offset
: The character position at which to split the Text node.

Returns

The Text node that was split from this node.

Throws

This method may throw a DOMException with one of the following code values:

INDEX_SIZE_ERR
: *offset* is negative or greater than the length of the Text or Comment node.

NO_MODIFICATION_ALLOWED_ERR
: The node is read-only and may not be modified.

Description

This method splits a Text node in two at the specified *offset*. The original Text node is modified so that it contains all text content up to, but not including, the character at position *offset*. A new Text node is created to contain all the characters from (and including) the position *offset* to the end of the string. This new Text node is the return value of the method. Additionally, if the original Text node has a parentNode, the new node is inserted into this parent node immediately after the original node.

The CDATASection interface inherits from Text, and this splitText( ) method can also be used with CDATASection nodes, in which case the newly created node is a CDATASection rather than a Text node.

See Also

Node.normalize( )

TreeWalker

DOM Level 2 Traversal

traverse a filtered document subtree

Properties

Node currentNode
: The current position of this TreeWalker and the node relative to which all of the TreeWalker traversal methods operate. This is the node most recently returned by one of those traversal methods or, if none of those methods have been called yet, it is the same as the root property.

  Note that this is a read/write property, and you can set it to any valid Document node—even one that is not a descendant of the original root node or one that would be rejected by the filters used by this TreeWalker. If you change the value of this property, the traversal methods operate relative to the new node you specify. Attempting to set this property to null throws a DOMException with a code of NOT_SUPPORTED_ERR.

`readonly boolean expandEntityReferences`
: This read-only property specifies whether this TreeWalker object expands entity references it encounters while traversing XML documents. The value of this property is set by the call to `Document.createTreeWalker( )`.

`readonly NodeFilter filter`
: The node filter function, if any, that was specified for this TreeWalker in the call to `Document.createTreeWalker( )`. If no node filter is in use, this property is `null`.

`readonly Node root`
: This read-only property specifies the `root` node at which the TreeWalker begins traversal. It is the initial value of the `currentNode` property and is specified in the call to `Document.createTreeWalker( )`.

`readonly unsigned long whatToShow`
: This read-only property is a set of bit flags (see "NodeFilter" for a list of valid flags) that specifies the types of Document nodes this TreeWalker will consider. If a bit is not set in this property, the corresponding node type will always be ignored by this TreeWalker. Note that the value of this property is specified in the call to `Document.createTreeWalker( )`.

Methods

`firstChild( )`
: Returns the first child of the current node that is not filtered out, or `null`.

`lastChild( )`
: Returns the last child of the current node that is not filtered out, or `null`.

`nextNode( )`
: Returns the next node (in document source order) that is not filtered out, or `null`.

`nextSibling( )`
: Returns the next sibling of the current node that is not filtered out, or `null`.

`parentNode( )`
: Returns the parent or nearest ancestor of the current node that is not filtered out, or `null`.

`previousNode( )`
: Returns the previous node (in document source order) that is not filtered out, or `null`.

`previousSibling( )`
: Returns the nearest previous sibling of the current node that is not filtered out, or `null`.

Description

A TreeWalker filters a specified document subtree and defines methods that allow programs to traverse the filtered tree (which may have a significantly different structure than the original document tree). Create a TreeWalker object with the `createTreeWalker( )` method of the Document object. Once a TreeWalker is created, you can use its `firstChild( )` and `nextSibling( )` methods to traverse the filtered subtree it represents in the same way that you might use the `firstChild` and `nextSibling` properties of the Node interface to traverse an unfiltered document tree.

A TreeWalker applies the same two filtration steps that a NodeIterator does. The various traversal methods defined by TreeWalker will return only nodes that pass both filters. First,

the node type must be one of the types specified by the whatToShow property. See "NodeFilter" for a list of constants that can be combined to specify the *whatToShow* argument to Document.createTreeWalker( ). Second, if the filter property is not null, each node that passes the whatToShow test is passed to the filter function specified by the filter property. If this function returns NodeFilter.FILTER_ACCEPT, the node is returned. If it returns NodeFilter.FILTER_REJECT, the node and all of its descendants are skipped by the TreeWalker (this differs from NodeIterator filtration, in which descendants are never automatically rejected). If the node filter function returns NodeFilter.FILTER_SKIP, the TreeWalker ignores the node but does consider its descendants.

Unlike NodeIterators, TreeWalkers are not modified when the underlying document is modified. The current node of a TreeWalker remains unchanged, even if that node is removed from the document. (And, in this case, the TreeWalker can be used to traverse the tree of deleted nodes, if any, that surround that current node.)

Example

```
// A NodeFilter that rejects <font> tags and any element with a
// class="sidebar" attribute and any descendants of such an element
var filter = function(n) {
    if (n.nodeName == "FONT") return NodeFilter.FILTER_SKIP;
    if (n.nodeType == Node.ELEMENT_NODE && n.className == "sidebar")
        return NodeFilter.FILTER_REJECT;
    return NodeFilter.FILTER_ACCEPT;
}

// Create a TreeWalker using the filter above
var tw = document.createTreeWalker(document.body,  // Walk HTML document body
                                   // Consider all nodes except comments
                                   ~NodeFilter.SHOW_COMMENT,
                                   filter,  // Use filter above
                                   false);  // Don't expand entity references

// Here's a recursive function that traverses a document using a TreeWalker
function traverse(tw) {
    // Remember the current node
    var currentNode = tw.currentNode;

    // Loop through the children of the current node of the TreeWalker
    for(var c = tw.firstChild(); c != null; c = tw.nextSibling()) {
        process(c);    // Do something to process the child
        traverse(tw);  // And recursively process its children
    }

    // Put the TreeWalker back in the state we found it in
    tw.currentNode = currentNode;
}
```

See Also

NodeFilter, NodeIterator; Chapter 17

Returned by: Document.createTreeWalker( )

TreeWalker.firstChild( )

DOM Level 2 Traversal

return the first child that is not filtered out

Synopsis

```
Node firstChild( );
```

Returns

The first child of the current node that is not filtered out, or `null` if there is no such child.

Description

This method sets `currentNode` to the first child of the current node that is not filtered out and returns that child. If there is no such child, it leaves `currentNode` unchanged and returns `null`.

TreeWalker.lastChild( )

DOM Level 2 Traversal

return the last child that is not filtered out

Synopsis

```
Node lastChild( );
```

Returns

The last child of the current node that is not filtered out, or `null` if there is no such child.

Description

This method sets `currentNode` to the last child of the current node that is not filtered out and returns that child. If there is no such child, it leaves `currentNode` unchanged and returns `null`.

TreeWalker.nextNode( )

DOM Level 2 Traversal

return the next node that is not filtered out

Synopsis

```
Node nextNode( );
```

Returns

The node that follows the current node in the document source and is not filtered out, or `null` if there is none.

Description

This method sets `currentNode` to the next node (in document source order) that is not filtered out and returns that node. If there is no such node, or if the search for the next

node takes the TreeWalker outside of the `root` subtree, `currentNode` remains unchanged and the method returns `null`.

Note that this method "flattens" the document tree structure and returns nodes in the order in which they appear in the document source. Calling `nextNode( )` may cause the current node to move down, sideways, or up the document tree. This type of flattening traversal can also be performed with `NodeIterator.nextNode( )`.

TreeWalker.nextSibling( )

DOM Level 2 Traversal

return the next sibling that is not filtered out

Synopsis

```
Node nextSibling( );
```

Returns

The next sibling of the current node that is not filtered out, or `null` if there is no such sibling.

Description

This method sets `currentNode` to the next sibling of the current node that is not filtered out and returns that sibling. If there is no such sibling, it leaves `currentNode` unchanged and returns `null`.

TreeWalker.parentNode( )

DOM Level 2 Traversal

return the nearest ancestor that is not filtered out

Synopsis

```
Node parentNode( );
```

Returns

The nearest ancestor of the current node that is not filtered out, or `null` if there is no such ancestor.

Description

This method sets `currentNode` to the nearest ancestor of the current node that is not filtered out and returns that ancestor. If there is no such ancestor, it leaves `currentNode` unchanged and returns `null`.

TreeWalker.previousNode( )

DOM Level 2 Traversal

return the previous node that is not filtered out

Synopsis

```
Node previousNode( );
```

Returns

The nearest node that precedes the current node in the document source and is not filtered out, or `null` if there is none.

Description

This method sets `currentNode` to the previous node (in document source order) that is not filtered out and returns that node. If there is no such node, or if the search for the previous node takes the TreeWalker outside of the `root` subtree, `currentNode` remains unchanged and the method returns `null`.

Note that this method "flattens" the document tree structure and returns nodes in the order in which they appear in the document source. Calling `previousNode( )` may cause the current node to move down, sideways, or up the document tree. This type of flattening traversal can also be performed with `NodeIterator.previousNode( )`.

TreeWalker.previousSibling( )

DOM Level 2 Traversal

return the previous sibling that is not filtered out

Synopsis

```
Node previousSibling( );
```

Returns

The previous sibling of the current node that is not filtered out, or `null` if there is no such sibling.

Description

This method sets `currentNode` to the closest previous sibling of the current node that is not filtered out and returns that sibling. If there is no such sibling, it leaves `currentNode` unchanged and returns `null`.

UIEvent

DOM Level 2 Events

details about user interface events

Event → UIEvent

Subinterfaces

MouseEvent

Properties

`readonly long detail`

A numeric detail about the event. For click, mousedown, and mouseup events (see "MouseEvent"), this field is the click count: 1 for a single-click, 2 for a double-click, 3 for a triple-click, and so on. For DOMActivate events, this field is 1 for a normal activation or 2 for a "hyperactivation," such as a double-click or **Shift-Enter** combination.

`readonly AbstractView view`

The window (the "view") in which the event was generated.

Methods

`initUIEvent( )`
: Initializes the properties of a newly created UIEvent object, including the properties inherited from the Event interface.

Description

The UIEvent interface is a subinterface of Event and defines the type of Event object passed to events of type DOMFocusIn, DOMFocusOut, and DOMActivate. These event types are not commonly used in web browsers, and what is more important about the UIEvent interface is that it is the parent interface of MouseEvent.

See Also

Event, MouseEvent; Chapter 19

UIEvent.initUIEvent( )

DOM Level 2 Events

initialize the properties of a UIEvent object

Synopsis

```
void initUIEvent(String typeArg,
                 boolean canBubbleArg,
                 boolean cancelableArg,
                 AbstractView viewArg,
                  long detailArg);
```

Arguments

typeArg
: The event type.

canBubbleArg
: Whether the event will bubble.

cancelableArg
: Whether the event may be canceled with `preventDefault( )`.

viewArg
: The window in which the event occurred.

detailArg
: The `detail` property for the event.

Description

This method initializes the `view` and `detail` properties of this UIEvent and also the `type`, `bubbles`, and `cancelable` properties inherited from the Event interface. This method may be called only on newly created UIEvent objects, before they have been passed to `EventTarget.dispatchEvent( )`.

ViewCSS

see AbstractView