HTML, CSS and JavaScript

[Nematrian website page: HTMLCSSJSTutorial, © Nematrian 2020]

Version Dated: 16 April 2020

In the pages set out below we introduce Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript, the three core components of most web pages.

In these pages, text used within HTML, CSS or JavaScript files is generally shown in `courier new` (i.e. a fixed space) font. The pages contain links to an extensive body of reference material explaining HTML, CSS and JavaScript in detail. We also provide a wide range of examples, which can help you understand better how HTML, CSS and JavaScript work. See below for further details on how to access these examples.

Hypertext Markup Language (HTML)

Cascading Style Sheets (CSS)

JavaScript

Appendices (reference material)

To access HTML, CSS or JavaScript examples please go to the webpage on www.nematrian.com that covers the specific feature you are seeking help with. More detailed examples (such as how to draw spinning3d shapes) are provided here.

HTML Tutorial

1. Introduction

[HTMLTutorialIntroduction]

Hypertext Markup Language (HTML) is one of the three main components of modern webpages, along with Cascading Style Sheets (CSS) and JavaScript. HTML indicates to the browser what elements should be included in the webpage (and in what order). CSS indicates how each element should be styled. JavaScript provides a means for webpage authors to manipulate these elements programmatically and in response to actions by the end user. Tutorials and reference material covering all three components are available here.

In these pages, we describe HTML further. Text used within HTML, CSS or JavaScript files is generally shown in `courier new` (i.e. a fixed space) font. The pages contain links to an extensive body of reference material explaining HTML, CSS and JavaScript in detail. We also provide a wide range of examples, which can help you understand better how HTML, CSS and JavaScript work. See below for further details on how to access these examples.

The concept of a markup language is explained further here. A document written in a markup language like HTML has parts that get rendered in the eventual output, but also parts that inform the rendering software how to interpret the remaining text. 'Rendering' here refers to the process of transforming the text document containing the HTML text into e.g. its visual representation on a screen.

The markup used by HTML includes tags, like `<p>…</p>`, to demarcate different HTML elements within the same webpage. In this case the `<p>` tag opens the relevant element and the `</p>` closes it. `<p>` elements are typically used to delimit paragraphs in HTML. HTML elements can be nested within other elements. Most elements can also be qualified by a range of attributes. For example, if we want to make the text within a `<p>` element appear red we can ascribe it a CSS style, along the lines of `<p style="color:red;">`.

Over time HTML has been refined. At the time of writing the latest version is HTML 5. Some aspects of earlier versions of HTML are no longer recognised in HTML 5 and some of these are noted where relevant.

Tutorial contents:

To access HTML, CSS or JavaScript examples please go to the webpage on www.nematrian.com that covers the specific feature you are seeking help with.

Disclaimer: Whilst we have made efforts to check the accuracy of the material in these pages, you should note that HTML, CSS and JavaScript are evolving languages. Information contained in this

document may therefore be inaccurate or out-of-date. You should not rely on the accuracy or fitness for purpose of any material that you obtain from the Nematrian website (or from its associated web services). If you need these results to be accurate or fit for purpose then you should seek independent corroboration of whatever you extract from the Nematrian website. Whilst using the site, users may be directed to the websites of other organisations, over which Nematrian may have no control and for which it takes no responsibility. A copy of the current Nematrian Web Services License Agreement can be viewed here.

2. Getting started with HTML

[HTMLTutorialGettingStarted]

As explained in HTML and other markup languages, there are various 'dialects' of HTML. This means that some examples of HTML may be understood by some browsers but rejected by others. The following text, when put into a text editor and saved with a .htm file extension, will usually successfully render a web page that says "Hello World (using HTML)" if the file is viewed in Microsoft Edge. Note that HTML largely ignores page breaks; if you want to include a page break in the text shown to the user then you need to add a `<br>` element (or a `<br />` element if you are using XHTML, which is a modern variant of HTML that involves a cross between classic HTML and XML).

```
<html>
   <body>
       Hello World (using HTML)
   </body>
</html>
```

However, strictly speaking an HTML document is supposed to start with a document type declaration, along the lines of e.g. `<!DOCTYPE html>` and a header along the lines of e.g. `<head><title>`*Document title*`</title></head>`. So, a better way to create the page shown above is as follows. We've added a comment into the document, using HTML comment tags. Comments are not displayed by the browser but can help to document the HTML source text.

```
<!DOCTYPE html>
<html>
   <head>
      <title>My Document</title>
   </head>
   <!-- Only the text in the body will appear in the browser -->
   <body>
       Hello World (Using HTML)
   </body>
</html>
```

Often, the `<html>` element also includes a `lang` attribute, as this can be important for accessibility applications (such as screen readers) and for search engines. With the `lang` attribute, the first two letters specify the language. If the language comes in various dialects then two more letters specify the dialect, e.g.:

```
<html lang="en-US">
```

3. Carriage returns and thematic break lines

[HTMLTutorialLineBreaks]

HTML markup largely ignores carriage returns (i.e. line breaks) within marked up files. Instead, if you want to insert a carriage return you need to insert a `<br>` tag.

By 'largely ignores' we mean that browsers do not render carriage returns as line breaks as such (except if certain styles apply to the element within which they appear, see e.g. the default formatting of the <pre> element). However, carriage returns do affect how HTML handles spaces at the end of the line and at the start of the following line. Leading spaces (i.e. ones on a line before any non-space characters) are typically ignored, as are trailing spaces (i.e. ones on a line after the last non-space character). However, browsers typically insert a 'breaking space' at the end of each line, which often then shows up as a single space. Multiple spaces one after the other are interpreted as a single space. To include more than one space in such instances you need to include a 'non-breaking space' as a special character, see here.

For example the following markup:

```
Hello (followed by two carriage returns)<br><br />
   Hello
     again
and again
```

creates the following output:

Hello (followed by two carriage returns)

Hello again and again

Webpage authors typically put a lot of effort into creating visually appealing material. One way of breaking up text is to insert a thematic break line or horizontal rule, i.e. a `<hr>` tag, which places a line across the window like:

---

4. Commenting

[HTMLTutorialCommenting]

It can be helpful to include comments in HTML documents that are not displayed but help readers of the underlying markup text you to understand what the HTML is trying to do. Comments in HTML take the form `<!--` *comment* `-->` and ignore line breaks within the opening and closing tags of the comment element.

For example, markup as follows:

```
<!-- material explaining how commenting in HTML
```

```
works-->
```

creates the following output (i.e. nothing):

5. Special characters

[HTMLTutorialSpecialCharacters]

The underlying markup of a webpage typically contains many more ampersand characters (i.e. `&`) than appear in the rendered output. This is because the `&` character is part of the way in which HTML marks up 'special' characters, i.e. ones that would otherwise be understood by HTML to relate to markup. In HTML, each special character is preceded by an ampersand, followed by the HTML markup name for that character followed by a semicolon. Perhaps the most common special characters are:

Special character	Meaning	HTML code
ampersand	&	`&`
space (technically a 'non-breaking' space)	(e.g. as in Hello again)	` ` (e.g. as in `Hello   again`)
less than sign	<	`<`
greater than sign	>	`>`
quotation mark	"	`"`
apostrophe	'	`'`

A fuller list of HTML special characters is available here.

6. Hyperlinks

[HTMLTutorialHyperlinks]

Many people associate web pages with hyperlinks, i.e. the ability to navigate from one page to another page. In HTML, hyperlinks (also called 'anchors') typically have the following sort of structure:

```
<a href="Pages/AboutNematrian.pdf";>text</a>
```

The *text* is what the user sees, the value of `href` is where the link points to. Points to note include:

(a) The 'text' material seen by the user can contain HTML, so can include e.g. images and formatted text
(b) The `href` value used here, i.e. "`Pages/AboutNematrian.pdf`" means that the link points to a webpage (or other resource) called "AboutNematrian.pdf" in the directory "Pages" (strictly speaking a subdirectory of the directory in which the source webpage resides, unless it e.g. starts with `http://` or `https://` or unless the document's <base> element, if any, defines a different base address to be used by relative uniform resource locators, i.e. 'URLs').

The above link renders as:

text

Groups of hyperlinks can be included in a <nav> element. For example, markup as follows:

```
<nav>
<a href="Introduction.aspx">Introduction</a> |
<a href="IntroductionSoftware.aspx">Software</a>
</nav>
```

creates the following output, involving 2 individual hyperlinks:

Introduction | Software

7. HTML elements and their attributes

[HTMLTutorialHTMLElements]

The basic building blocks of HTML are elements (also called tags). A list of recognised elements is shown here. Some HTML elements, like the hyperlinks in HTMLTutorialHyperlinks, are by default differentiated from the text around them. The most general way of formatting text (capable of altering any of the default formatting of any visible HTML element) involves use of Cascading Style Sheets (CSS) or in-file or in-line equivalents, see Nematrian's CSS Tutorial. In-line CSS involves assigning a specific style attribute to a given element. Other sorts of attributes can also be assigned to different sorts of elements. A list of recognised attributes is shown here. The exact range of attributes valid for a specific element type does vary; see individual elements or attributes for further details.

Many other elements are also by default differentiated from the text around them or exist primarily to facilitate this sort of differentiation. Examples include:

HTML element	Normally used for e.g.	example
<address>	Contact information for the author or owner of a document	Mr. Smith, 1, George Street, Georgetown
<b>	Bold text	**1, George Street, Georgetown**
<blockquote>	Section that is quoted from another source	1, George Street, Georgetown
<cite>	Title of a work	Systemic Risk
<code>	A piece of computer code	`var x = 2.0;`
<del>	Indicates text deleted from a document	a~~bc~~
<dfn>	Defining instance of a term	*HTML*, a markup language
	Emphasised text (often used to italicise text, but ideally this should be done using CSS, as emphasis does not need to involve italics)	*HTML*, a markup language
<footer>	Footer for a document or section	HTML

<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	HTML headings	**Header 1** **Header 2** **Header 3** **Header 4** **Header 5** **Header 6**
<header>	Header for a document or section	HTML
<i>	A part of text in an alternate voice or mood	*HTML*, a markup language
<ins>	Indicates text added to a document	def
<kbd>	Keyboard input	`text representing keyboard input`
<mark>	Marked/highlighted text	text to highlight
<pre>	Preformatted text	`preformatted text (in fixed -width font that also preserves spaces and line breaks`
<q>	Short quotation	Mary had a little lamb
<s>	Text that is no longer correct	~~abc~~
<samp>	Sample output from a computer program	`output`
<small>	Smaller text	1, George Street, Georgetown
	Defines more important text, commonly used as another way of highlighting text or making it bold	**Mary had a little lamb**
<sub>	Subscripted text	A_1
<summary>	Heading for a <details> element	Heading
<sup>	Superscripted text	A^1
<time>	Date / time (N.B. isn't normally differentiated from standard text in most modern browsers)	10:00
<u>	Text that should be stylistically different from normal text, commonly used for underlining	abc
<var>	Variable	*Param1*
<wbr>	Posible line-break, the Word Break Opportunity tag specifies where in a text it would be ok to add a line-break	

Some HTML elements are no longer supported in HTML5. Although they often work in browsers you should ideally use CSS instead. These include:

HTML element	Normally used for e.g.	example
<big>	Big text	1, George Street, Georgetown
	Centred text	abc
<font> as in e.g. `<font color="green">`	Green text	green text
<strike>	Strikethrough text, not supported in HTML 5 (instead use <del> or <s>)	~~abc~~
<tt>	Teletype text	abc

Some HTML tags differentiate content, but primarily to assist the browser's understanding of that content, e.g.:

HTML element	Normally used for e.g.	example
<abbr> as in e.g. `<abbr title="Mister">`	Abbreviation or acronym	Mr.
<data> as in e.g. `<data value="1011">`	Links content with a machine-readable translation	Apple

Some HTML tags demarcate content so that the material can be segmented up more effectively, or assigned different formatting styles, e.g.:

HTML element	Normally used for e.g.	example
<article>	Article	**Title** Material
<aside>	Content aside from the page content	**Title** Material
<details>	Additional details that a user can view or hide	**Title** Material
	Section in a document	a single piece of content
<main>	Main content of a document	main text
<p>	Paragraph (by default has space added above and below the text)	a paragraph
<section>	Section in a document	a section
	Section in a document	a section (span)

A summary of the default styles applied to each HTML element is set out here.

8. Browser feature detection

[HTMLTutorialFeatureDetection]

Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript form the main elements of modern webpages. However, different browsers do not always interpret these webpage components in entirely consistent ways.

There are two main ways in which this occurs:

(a) HTML, CSS and JavaScript are evolving through time, particularly as new ways of interacting with webpages are developed. Most of their core components are understood by essentially all browsers. However, some newer features may only work on some browsers. Some may be released in 'beta' or 'test' form, but may eventually be dropped in any finalised updates.

(b) Webpages are nowadays accessed across a wide range of formats. These formats can take different physical forms. Even when they involve a 'traditional' screen-based format, the screens can come in many different sizes and resolutions (e.g. a PC-based screen is typically larger, and easier to resize, than a mobile phone-based screen). This makes it desirable to alter the way in which material is displayed depending on the format involved.

Historically, webpage developers solved this problem using 'browser detection'. In this approach, the developer would include in the webpage (or on the server delivering the webpage to the user) some means of detecting which browser was being used to access the webpage. This had two main weaknesses. First, there are now many different browser providers most of whom also have many versions of their browsers available. This made it very difficult for developers to keep up with the changing browser scene. Second, a 'browser detection' approach fails to address (b) above. The same browser can run on multiple devices; if the devices themselves have different characteristics then these won't be captured merely by identifying the browser being used.

Nowadays, the trend is to use 'feature detection'. In this approach, the developer includes elements or other components in the webpage that identify if the browser and device being used to access the webpage supports a specific feature. The output is then optimised bearing in mind whether the feature is available.

Sometimes this type of functionality is explicitly built into HTML. For example, the media file formats recognised by HTML <audio> and <video> elements vary by browser. These HTML elements specifically allow developers to refer to more than one media source, depending on the format involved. The browser being used can then select whichever source it recognises. If it doesn't recognise any (or if it is not new enough to recognise <audio> or <video> elements), fallback text will be displayed. The CSS @media rule can also be used in a way that allows the developer to alter the style used by an element to reflect the media in which the page is being viewed (e.g. the width or height of the device).

At other times, feature detection needs to be coded using JavaScript. The typical approach is to identify whether a feature is supported by the browser and then to adjust output formatting accordingly. However, it is not always easy to identify whether a specific feature is supported by the browser. Possible methods include:

- Use the `hasFeature` method to determine whether the JavaScript Document Object Model (DOM) implementation supports the relevant feature
- Search for DOM objects, properties or methods associated with the feature

- Attempt to create an object that should have the feature and if creation is successful then test whether it does support the feature

Unfortunately, the `hasFeature` method is not well supported by several browsers and its use is often not recommended. The Nematrian website includes functions for many JavaScript features that can assist in checking whether the feature is being supported by the browser being used at the time. See pages on individual features for further details.

CSS Tutorial

1. Introduction

[CSSTutorialIntroduction]

Cascading Style Sheets (CSS) is one of the three main components of modern webpages, along with Hypertext Markup Language (HTML) and JavaScript. HTML indicates to the browser what elements should be included on the page (and in what order). CSS indicates how each should be styled. JavaScript provides a means for webpage authors to manipulate these elements programmatically and in response to actions by the end user. Tutorials and reference material covering all three components are available here.

In these pages, we describe CSS further. Text used within HTML, CSS or JavaScript files is generally shown in `courier new` (i.e. a fixed space) font. The pages contain links to an extensive body of reference material explaining HTML, CSS and JavaScript in detail. We also provide a wide range of examples, which can help you understand better how HTML, CSS and JavaScript work. See below for further details on how to access these examples.

CSS instructions can be:

(a) included within an individual HTML element (as part of the mark-up relating to that element), i.e. as 'in-line' CSS
(b) included in the HTML file where the relevant element(s) are located, but not directly within the elements concerned, i.e. as 'in-file' CSS
(c) included in external CSS files, i.e. as 'external' CSS, with a HTML <link> element used to indicate where any such CSS files applicable to a given HTML file are located.

The style attributes of an HTML element can also be altered by JavaScript 'on the fly', e.g. after the page has initially loaded or in response to specific user actions such as clicking a button.

CSS styles typically operate according to a hierarchy, with any JavaScript overrides taking precedence over any CSS styles present when the page is initially loaded but otherwise in-line CSS taking precedence over in-file CSS and in-file CSS taking precedence over external CSS (unless the 'important' characteristic is included in the style statement). In-file CSS is contained in a <style> element. If there is more than one such element then later ones take precedence over earlier ones.

Older versions of HTML (e.g. HTML 4) require <style> elements to be in the `<head>` of the HTML file, although most browsers currently seem to accept them even if they appear in the `<body>`. In theory, the latest HTML version at the time of writing (HTML 5) has the concept of a 'scoped' attribute (e.g. `<style scoped>`) which should allow you to apply different `<style>` elements to different parts of the webpage (which could then legitimately appear in the `<body>` element), but not all browsers currently seem to cater for this aspect of HTML 5.

External style sheets are referenced using a <link> element, which goes inside the `<head>` section. This type of link element has a form such as:

```
<link rel="stylesheet" type="text/css" href="mystyle.css">.
```

External style sheets can be created in any text editor, should not contain any HTML tags (elements) and should be saved with a `.css` extension.

In-file and external CSS are typically set out in the form of 'rule-sets'. A rule set involves a selector and a declaration block. The selector points to the type of HTML element to which the style applies, whilst the declaration block contains one or more style declarations separated by semicolons. Each declaration involves a CSS property name, followed by a colon, followed by the value assigned to the property.

For example, the style rule

```
h3 {color: blue; text-align: center;}
```

has a selector which is `h3` and a declaration block which is `{color: blue; text-align: center;}`. It tells the browser that any `<h3>` element (to which the rule applies) should be centre-aligned and appear in blue. As with HTML, line breaks and multiple spaces are ignored.

Other types of selectors are introduced here and covered in more detail here.

In-line CSS rule-sets involve the style attribute (and do not include a selector or the curly brackets / braces included in in-file or external CSS), e.g. they involve setting the element's style attribute along the lines of: `style = "color: red";`.

Comments in CSS start with `/*` and end with `*/` and can span multiple lines.

Over time CSS has been refined. At the time of writing the latest version is CSS3. Features in CSS1 and CSS2 can typically still be used in CSS3.

Tutorial content

To access HTML, CSS or JavaScript examples please go to the webpage on www.nematrian.com that covers the specific feature you are seeking help with.

2. Selectors

[CSSTutorialSelectors]

CSS is typically set out in the form of 'rule-sets', which involve a selector and a declaration block. Usually CSS is applied to types of elements. For example, the style rule

```
h3 {color: blue; text-align: center;}
```

has a selector which is `h3` and a declaration block which is `{color: blue; text-align: center;}`. It tells the browser that any `<h3>` element (to which the rule applies) should be centre-aligned and appear in blue. As with HTML, line breaks and multiple spaces are ignored.

However, within HTML you can also define classes of elements with common formatting styles using the element's class attribute. For example, the style rule

```
.center {color: red; text-align: center}
```

would indicate that any element with a class attribute equal to `center` should be centre-aligned and appear in red.

You can also apply CSS to elements of a specific type *and* class. For example, the style rule

```
h3.center {color: green;}
```

would indicate that `<h3>` elements that have their class attribute equal to `center` should be green.

In-file CSS can also be applied to individual elements, if the id attribute of the HTML element has been set (the id attribute should be unique within any given page). If you want to use this type of CSS then precede the id value by a hash (`#`) character.

For example, the style rule

```
#para1 {color: yellow}
```

would be applied to the HTML element with id equal to `para1` (provided there is such an element) and it would appear yellow (unless overridden by a later style rule).

You can also group together rules for elements with the same style definitions, separating each selector with a comma. For example,

```
h1 {color: red;}
h2 {color: red;}
h3 {color: red;}
```

can be grouped together as follows to minimise code and make it easier to follow:

```
h1, h2, h3 {color: red;}
```

More general ways of identifying CSS selectors are set out here.

3. Hints and further information

[CSSTutorialHints]

CSS Values

In CSS, if you are using values that have units, e.g. applying values that are to be interpreted as CSS lengths (e.g. setting the size of an element's left margin using e.g. `margin-left: 20px`) then you should not include a space between the value (here `20`) and the unit (here `px`) as otherwise the style may be ignored.

There are several ways of defining lengths in CSS. There are also specific conventions used when defining CSS times, CSS angles and CSS colours.

Hierarchy in CSS style rules

If you have two or more style rules that would otherwise apply to a specific attribute of a specific element then the hierarchy rules are that:

- More specific rules override more general ones. Specificity is defined based on how many IDs, classes and element names are involved as well as by whether there is an `!important` declaration.
- When even these do not differentiate between styles then whichever one appears last is the one that is applied.

For example, without the `!important` flag, `<h3>` elements using the following styles would appear green (as the green style rule is after the red one), but with the `!important` flag it is the red one that applies in this instance:

```
h3 {color: red !important}
h3 {color: green}
```

Setting the CSS style of the whole page

The style of the whole page can be set by a style rule such as:

```
body {background-color: lightblue;}
```

Multi-valued CSS properties

Some CSS properties take several values. For example, many HTML elements are deemed to have 4 sides (top, right, bottom and left) and there are conventions on how to define properties that encompass all four sides simultaneously, see here.

More generally, some CSS properties are shorthand properties that set several other more granular properties at the same time.

Animation and other more sophisticated features

CSS has developed over the years and it is now possible to create relatively sophisticated animation effects using the CSS @keyframes rule, without needing to implement these animations using JavaScript. It is also possible to apply different styles depending on the device being used to render the material, using the CSS @media rule. Material can be automatically added before or after HTML elements using CSS 'pseudo-properties', such as the content pseudo-property.

Styling of hyperlinks

Links can be styled differently depending on what state they are:

Link state	Description
a:link	Normal, unvisited link
a:visited	Link that user has visited
a:hover	Link when the user moves a mouse over it
a:active	Link at the moment it is clicked

Advanced table formatting

Zebra-striped tables can be implemented using the nth-child selector, e.g.:

```
tr:nth-child(even) {back-ground-color: #f2f2f2;}
```

To make a table responsive (i.e. to display a horizontal scroll bar if the screen is too small to display in full) you can add a container element with `overflow-x:auto`, e.g.:

```
<div style="overflow-x:auto;"><table> … </table></div>
```

JavaScript Tutorial

1. Introduction

[JavaScriptTutorialIntroduction]

JavaScript is one of the three main components of modern webpages, along with Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS). HTML indicates to the browser what elements should be included on the page (and in what order). CSS indicates how each should be styled. JavaScript provides a means for webpage authors to manipulate these elements programmatically and in response to actions by the end user. Tutorials and reference material covering all three components are available here.

In these pages, we describe JavaScript further. Text used within HTML, CSS or JavaScript files is generally shown in `courier new` (i.e. a fixed space) font. The pages contain links to an extensive body of reference material explaining HTML, CSS and JavaScript in detail. We also provide a wide range of examples, which can help you understand better how HTML, CSS and JavaScript work. See below for further details on how to access these examples.

JavaScript can be added to a webpage in one of three ways (somewhat akin to how CSS can be added to a webpage):

(a) By including it within an individual HTML event attribute. This typically involves only very small JavaScript statements.
(b) Within separate <script> elements in the HTML
(c) In external script files (these involve including in the HTML a <script> element with its src attribute set to the relevant script file name).

A simple example of JavaScript involves the use of the `document.write` method. For example, the following HTML text would return a web page the first line of which says "Hello World (using HTML)" followed by a line break and a second line saying "Hello World (using HTML)". Script elements are typically executed in the order in which they appear when the page is first loaded. In this case the script cause the browser to add some more text to the web page.

```
<html>
   <body>
       Hello World (using HTML)<br>
      <script>
         <!--
            document.write("<br>Hello World (using Javascript)")
         //-->
      </script>
   </body>
</html>
```

More sophisticated approaches can alter individual HTML elements rather than merely adding to the end of the document or can react to events such as the clicking of a button. For example, the following HTML text returns a web page with two lines, the first being "Hello World (using HTML)" and the second line being "*and using JavaScript*".

```
<!DOCTYPE html>
```

```
<html>
<head></head>
<body>
Hello World (using HTML)<br>
<em id="Added"></em>

<script>
document.getElementById("Added").innerHTML="and using JavaScript"
  // Adds text to the element with id="Added"
</script>

</body>
</html>
```

Note: we are here taking advantage of the execution of script commands when the page first loads. A more complicated (but more general way) of achieving the same result would be to add an 'event listener' that is triggered when the page loads and to have a function associated with this event listener that alters (here adds) the text in the desired manner when the event happens. By attaching the function to a different event, e.g. one triggered when the user clicks on an element then the a more responsive webpage can be created.

JavaScript comments

When writing computer software, it often helps to add explanatory comments. In JavaScript, a single line comment is indicated by "*code* // *text*" where the code is still executed, but the text is ignored by the Browser.

Any text between "/*" and "*/" (not in quotes) including line breaks is also ignored, allowing authors to create multi-line comments. These tend to be used for formal documentation, e.g. material at the start of each function that describes what the function does.

Tutorial contents:

To access HTML, CSS or JavaScript examples please go to the webpage on www.nematrian.com that covers the specific feature you are seeking help with.

Nematrian may have no control and for which it takes no responsibility. A copy of the current Nematrian Web Services License Agreement can be viewed here.

2. Variables

[JavaScriptTutorialVariables]

A variable in JavaScript is defined by a command such as:

```
var x;
```

If you want to set a variable to a value when it is first defined then you generally use the assignment operator within this definition, e.g.:

```
var x = 10;
```

JavaScript recognises the following types of 'primitive' variables:

- String variables
- Number variables
- Date variables
- Boolean variables

Variables can also be objects and arrays (and for some string manipulation purposes, regular expressions). In JavaScript, an array is a special type of object that is indexed along the lines of `a[0]`, `a[1]`…. Arrays can consist of other objects, including other arrays.

Several variables can be defined in the same statement, with each one separated by a comma, e.g.:

```
var x = 10, y = 15, z = 20;
```

Variables that have not yet been defined a value have their value as `undefined`.

If you redefine a variable, it retains its previous value. For example, after the statements

```
var x = 10;
var x;
```

the variable `x` still has the value `10`.

Variables are manipulated using operators and functions. For example, numbers can be added together using the addition operator or functions can be applied to them, e.g.:

```
var x = 0.1 + 0.2;
function sinsquared(x) {
  var a;
  a = Math.pow(Math.sin(x),2);
  return a;
}
var y = sinsquared(0.3);
```

JavaScript variable names (i.e. identifiers) follow certain rules:

- They can contain only letters, digits, underscores and dollar signs
- They must typically begin with a letter (in some cases they can also begin with a $ or an _ character)
- The names are case sensitive (i.e. a and A are different names)
- They cannot be reserved words, such as those used in JavaScript statement construction)

An important concept in programming is the *scope* of a variable (or more precisely of its name). This is the part of the code within which the relevant variable is accessible via its name. If code is segmented into blocks then it is often desirable to use a similar variable name in different blocks but for the names to then be associated with different variables depending on the block in question. The scope of a JavaScript can be *local* or *global*. Variables defined inside functions are local to that function, whilst those defined outside functions are global in scope. Local variables are deleted when the function completes, while global variables remain available until the user closes the browser window or tab within which the page has been loaded. This means that they are available to new pages loaded into the same browser window. Function arguments work in the same manner as local variables inside a function.

String variables

Strings consist of a series of consecutive characters, e.g.

```
var x = "Cat";
```

A string technically consists of a series (an 'array', except that a JavaScript array is a specific type of variable) of characters, which is zero-indexed. So, if we assigned `x` the value of `"Cat"` then `x[0]` would be `"C"`, `x[1]` would be `"a"`, etc.

Further details on the methods and properties supported by string variables are set out in JavaScript Tutorial: Strings.

Regular expressions

Some string methods and properties involve 'regular expressions'. These take the form:

/pattern/modifiers

e.g.:

```
var x = /nematrian/i;
```

Further details on the methods and properties supported by regular expressions variables are set out in JavaScript Tutorial: Regular Expressions.

Numbers (and mathematical manipulations)

JavaScript has only one type of number (in contrast to, e.g. Visual Basic, which differentiates between e.g. integers, floating point numbers and double precision numbers). Numbers can be written with or without decimal points and/or with or without (scientific) exponents), e.g.

```
var x = 4.1;     // With a decimal point
var y = 4;       // Without a decimal point
```

```
var p = 135e6   // Means 135000000
var q = 13.5e-3 // Means 0.0135
```

Further details on the methods and properties supported by numbers and by the Math object (which can be used to carry out mathematical manipulations) are set out in JavaScript Tutorial: Number variables and mathematical functions.

Dates

Date variables are objects and contain dates and times. They can be instantiated in 4 ways:

```
var d1 = new Date();            // An as yet undefined date
var d2 = new Date(milliseconds);  // See below
var d3 = new Date(dateString);    // See below
var d4 = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

Here *milliseconds* refers to the number of milliseconds since 1 January 1970 00:00:00. A *dateString* is a piece of text that the browser can recognise as representing a date.

Further details on the methods and properties supported by numbers and by the Math object (which can be used to carry out mathematical manipulations) are set out in JavaScript Tutorial: Dates.

Booleans

Boolean variables take one of two values, `true` or `false`. They are instantiated by a statement such as:

```
var b = true;
```

You can use the `Boolean()` function to identify whether an expression is true or false, although it is simpler just to use operators that return Boolean outputs, e.g. `Boolean(2 > 1)`, `(2 > 1)` or even `2 > 1` all return true.

Further details on the methods and properties supported by Boolean variables are shown in JavaScript Tutorial: Booleans.

Arrays

Arrays contain multiple (indexed) values in a single variable. Array indices are zero-based, i.e. the first element of the array has as its index 0, the second 1 etc. They are instantiated by statements such as:

```
var a = ["France", "Germany"];
var b = [1, 2, 5, 4];
```

It is worth noting that elements of arrays can themselves be arrays since technically an array is a specific type of object.

Further details on the methods and properties supported by arrays (and some of the subtleties that arise if you want to copy them) are set out in JavaScript Tutorial: Arrays.

Objects

JavaScript objects are containers that contain *properties* and *methods*. For example, a statement such as:

```
var person = {title:"Mr", surname:"Smith", age:30}
```

creates an object that has three properties, i.e. name-value, pairs that in this instance characterise (some of the features of) a person.

Object properties can be accessed in two ways, either here e.g. `person.title` or `person["title"]` (both of which in this instance would return a value of `"Mr"`). An array is a specific type of object with the property names indexed from 0 up to the length of the array less 1 (and hence elements of arrays can themselves be arrays or other sorts of objects).

Object methods are functions that can be applied to objects. They are technically also property-like in nature, i.e. again come in name-value pairs, but with the 'name' being a function name (with parameter definitions if necessary) and the 'value' being the JavaScript function script associated with that function, see JavaScript Tutorial: Objects.

A special type of object is the Error object, which is used for error handling.

3. Statements

[JavaScriptTutorialStatements]

JavaScript statements identify instructions that are executed by the web browser. For example, the following statement tells the browser to write "Hello World" inside an HTML statement with the id attribute = `"element"`:

```
document.getElementById("element").innerHTML = "Hello World"
```

The same result can be achieved using several separate statements, e.g.:

```
var d = document.getElementById("element");
var x = "Hello";
var y = " World";
var z = x + y;
d.innerHTML = z;
```

Statements are separated by semicolons and multiple statements are allowed on one line. JavaScript ignores multiple spaces (except in strings, i.e. within quotation marks). A common good practice is to put spaces around operators (e.g. =, + , …). Very long lines of code are also often frowned upon, and are usually broken after an operator.

Statements can (and often are) grouped together in code blocks, inside curly brackets, i.e. { … }. A particularly important example of the use of code blocks involves functions, which provide a means of executing on demand one or more statements, e.g.:

```
function func() {
      document.getElementById("element").innerHTML = "Hello";
}
```

Statements often start with a statement identifier. These are reserved words which cannot be used as variable names or for other purposes. A list of statement reserved words recognised by JavaScript is shown here. They include: `break`, `continue`, `do`, `for`, `if`, `return`, `switch`, `throw`, `try`, `catch`, `var` and `while`.

Most JavaScript programs contain many statements, which are executed one by one in the order in which they are written except when statement flow control is adjusted using statements such as `for`, `if` or `while`.

4. Functions

[JavaScriptTutorialFunctions]

A JavaScript function is a block of JavaScript code that can be executed as a discrete unit. It involves a function statement along the lines of e.g.:

```
function func() {
      document.getElementById("element").innerHTML = "Hello";
}
```

Function definitions can include parameters (separated by a comma if more than one parameter), e.g. the following (if passed a string variable) would allow any text to be inserted in the relevant element's innerHTML.

```
function func2(x) {
      document.getElementById("element").innerHTML = x;
}
```

Such a function would be invoked by JavaScript such as `func2("Hello World")`.

Functions are much like procedures or subroutines in other programming languages. The code inside the curly brackets executes when the function is invoked. This can happen when an event occurs, when the function is called from JavaScript code or sometimes when it is self-invoked. If a function includes a return statement then the function will stop executing and will return the value identified by the function's return statement. The function (technically, a special type of object) can be distinguished from the act of invoking it. The `()` operator invokes the function, e.g. in the above `func` refers to the function object, but `func()` will invoke the function itself.

The function parameters are the names listed in the function definition (i.e. the `x` in the definition of `func2`). Function arguments are the values received by the function (i.e. assigned to the function parameters) when it is invoked.

Function names can contain letters, digits, underscores and dollar signs (the same rules as apply to variable naming applies to function naming). Wherever a variable can be used, a valid function call evaluating to the same value can also be used.

5. Event handling

[JavaScriptTutorialEventHandling]

A responsive website needs to respond to users when the users act in specific ways, e.g. loading a page, clicking on a button, moving a mouse around a document window etc. JavaScript, like many other modern more sophisticated general-purpose programming languages, includes the concept of *events*. These assign specific functions to specific events, with the functions being invoked if/when the event occurs.

Event handling linked to individual elements, such as what happens when someone clicks on an element, is often implemented by assigning a specific function to the event attribute of that element, see here.

Global events (not linked to specific HTML elements), such as those triggered by loading the page, are typically implemented by using e.g. the document.addEventListener method, e.g.:

```
document.addEventListener('load', addtext());
```

6. The Document Object Model (DOM)

[JavaScriptTutorialDOM]

The JavaScript HTML Document Object Model ('DOM') provides a way for JavaScript to access all elements of an HTML webpage. Fuller details of the DOM are given here. There is an associated Browser Object Model (BOM), details of which are given here.

The browser creates a DOM (i.e. a `document` object) for a page when the page is first loaded. This has a tree structure, with the root element (or 'node') being the page's <html> element. The root element then has two sub-elements (or 'nodes'), i.e. the <head> element and the <body> element.

The <head> element will in turn often include as one of its 'child' nodes a <title> element. The <body> element contains the main body of the webpage and will typically contain many different elements, some nested within others. JavaScript can change all the elements (including all their attributes), can add new elements or remove existing ones, can react to all existing events and create new ones.

Formally, the DOM is a W3C (World Wide Web Consortium) standard. It has the following aim, according to W3C: "*The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of a document.*" It is formally subdivided into three parts: (a) the Core DOM for all document types, (b) the XML DOM for XML documents, and (c) the HTML DOM for HTML documents. It adopts an object-orientated programming approach, with the HTML elements being *objects* which have *properties* and to which can be applied *methods*. The elements can also trigger *events*.

A common way of accessing an element is to assign it an id (i.e. set its id attribute object to a prespecified value). The element can then be identified in Javascript by applying the `getElementById` method to the document object, returning an object corresponding to the element. Its properties (e.g. its innerHTML property, which is the text within an element) can be set by assigning values to the relevant property of the element object. For example, the following example returns a web page that says "hello".

```
<html>
<body>
```

```
<div id="example"></div>
<script>document.getElementById("example").innerHTML =
"hello"</script>
</body>
</html>
```

Common ways of accessing the DOM include:

Aim	Example JavaScript	Description
Finding / accessing elements	`document.getElementById(`*id*`)`	Returns object corresponding to element with this id attribute
	`document.getElementsByTagName(`*name*`)`	Returns collection of elements by tag name (i.e. by type of element)
	`document.getElementsByClassName(`*name*`)`	Returns collection of elements by class name
	`document.querySelectorAll(`*CSSSelector*`)`	Returns collection of elements by CSSSelector
Changing elements	*element*`.innerHTML = `*HTMLcontent*	Change inner HTML of element
	element.attribute = *value*	Change attribute value of element (value needs to be valid for that attribute)
	element`.setAttribute(`*attribute*`, `*value*`)`	Change attribute value for a given element
Adding and deleting elements	`document.createElement(`*element*`)`	Creates HTML element
	`document.appendChildElement(`*element*`)`	Add HTML element
	`document.removeChildElement(`*element*`)`	Remove HTML element
	`document.replaceChildElement(`*element*`)`	Remove HTML element
	`document.write(`*element*`)`	Write directly to HTML output

Other points to note about the DOM include:

(a) The DOM uses the idea of nodes and a node tree. This involves a tree structure where each branching point is a separate node. So, nodes belong to (are children of) just one other node (their parent) back to the root node (which in the DOM is the `document` object)
(b) HTML elements are 'element' nodes, the attributes of these elements are 'attribute' nodes, the text within HTML elements are 'text' nodes and comments are 'comment' nodes
(c) A NodeList object represents a set of nodes, e.g. an HTML element's collection of child nodes. These will be indexed and each node within the NodeList can then be associated with another NodeList (its children)
(d) The HTML document, once it has been loaded into the web browser, is formally part of the corresponding Window object and can therefore be accessed via `window.document`
(e) The DOM supports a range of (own) methods and properties, see here.
(f) HTML elements ('nodes') within the DOM also support a range of more generic methods and properties, see here. These also apply to the `document` object itself but do not always make much sense when applied in this manner.

(g) HTML element attributes are represented by an Attr object. These are always children of a specific HTML element. The properties and methods that apply to Attr objects are shown here.
(h) A NamedNodeMap object represents an unordered collection of nodes, e.g. the set of attributes assigned to a given HTML element. Properties and methods that apply to NamedNodeMap objects are shown here.
(i) The DOM object and its components can be thought of as an example of an XML document. XML documents have several methods and properties not otherwise covered in the above (such as XMLHTTPRequest, which can be used to send, request and receive data from the server once a webpage has loaded), see here.

Further details are set out in the following pages and in links within them:

1. DOM own properties and methods
2. HTML Element objects: Properties and Methods
3. HTML Attribute objects: Properties and Methods
4. NamedNodeMap objects: Properties and Methods
5. Event objects: Properties and Methods
6. MouseEvent objects: Properties and Methods
7. KeyboardEvent objects: Properties and Methods
8. HashChangeEvent objects: Properties and Methods
9. PageTransitionEvent objects: Properties and Methods
10. FocusEvent objects: Properties and Methods
11. AnimationEvent objects: Properties and Methods
12. TransitionEvent objects: Properties and Methods
13. WheelEvent objects: Properties and Methods
14. TouchEvent objects: Properties and Methods

I. Style objects: Properties and Methods
II. Creating and Accessing HTML Elements in JavaScript
III. Standard HTML DOM properties and methods
IV. The JavaScript BOM (Browser Object Model)
V. The JavaScript XML DOM

7. Miscellaneous

[JavaScriptTutorialMiscellaneous]

We set out below some further comments on JavaScript that may help developers.

JavaScript syntax:

- Statements are separated by semicolons, e.g. `var x, y, z;  x = 3;`
- The language consists of values, operators, expressions, keywords and comments
- Values can be literals (fixed values) or variables.
- Number literals are written with or without decimal points, e.g. `100` or `10.1` (commas should not be used as the decimal separator)
- String literals are text, written within double or single quotes, e.g. `x = "35"; y = 'a';`
- Variables are declared using the var keyword, e.g. `var x; var y = 10;`
- The equal sign is used to assign a value to a variable, e.g. `x = 3;`, i.e. it is the assignment operator

- An expression is a combination of values, variables and operators
- String concatenation is achieved by `+`, e.g. `"a" + "  " + "b"` evaluates to `"a  b"`
- Comments are noted by text after double slashes "`//`" or between `/*` and `*/`
- Identifiers are used to name variables. The first character must be a letter, an underscore character ("`_`") or a dollar sign ("`$`")
- JavaScript is case sensitive, e.g. `firstName`, `FirstName` and `firstname` will be three different variables
- JavaScript typically uses 'lower camel case' when joining multiple words together, i.e. first letter of first word is lower case, first letter of each subsequent word is upper case, e.g. "`firstName`". Some methods and properties specifically use this convention, and because JavaScript is case sensitive it means that using a different convention will result in an incorrect (or failed) evaluation. Developers often adopt the same convention when naming variables.
- JavaScript uses the Unicode character set

Displaying (text / numerical) data using JavaScript:

- Text and values can be inserted into an HTML element, using e.g. the `innerHTML` property
- We can write to the browser output stream, using e.g. `document.write()` (this is useful for testing purposes)
- We can write to an alert box, using e.g. `window.alert()`
- We can write into the browser console, using e.g. `console.log()`

Location of any JavaScript within HTML webpages:

- HTML in-file JavaScript needs to be inserted in script elements.
- Any number of scripts can be included in an HTML document and can be placed in either the <body> or the <head> section

Global methods and properties:

JavaScript has some *global* properties and functions that can be used with all built-in JavaScript objects. These include:

Global properties:

Property	Description	More
`Infinity`	A number that is positive infinity (`-Infinity` refers to a number that is negative infinity)	Here
`NaN`	Not-a-Number result of a numeric calculation	Here
`undefined`	Indicates variable has not been assigned a value	Here

Global methods:

These are perhaps best referred to as global 'functions' since they are called globally.

Method / Function	Description	More
`Boolean()`	Converts an object's value to a Boolean (i.e. `true` or `false`)	Here
`decodeURI()`	Decodes URI	Here
`decodeURIComponent()`	Decodes URI component	Here

encodeURI()	Encodes URI	Here
encodeURIComponent()	Encodes URI component	Here
escape()	Depreciated. Use encodeURI() or encodeURIComponent() instead	Here
eval()	Evaluates a string as if it were script code	Here
isFinite()	Returns true if finite, otherwise false	Here
isNaN()	Returns true if NaN, otherwise false	Here
Number()	Converts an object's value to a number	Here
parseFloat()	Parses a string and returns a floating-point number	Here
parseInt()	Parses a string and returns an integer	Here
String()	Converts an object's value to a string	Here
unescape()	Depreciated. Use decodeURI() or decodeURIComponent() instead	Here

Properties shared by multiple JavaScript objects:

JavaScript includes some properties that can be applied to any object and can allow the user to alter the methods and properties applicable to the object. These include:

Property	Description	More
constructor	Returns object's constructor function	Here
length	Returns the length of the object	Here
prototype	Allows author to add properties and methods to an object	Here

Conversions between variable types:

An area of JavaScript that can be confusing (and hence can lead to errors) is the handling of conversions between different variable types. JavaScript is 'weakly typed', i.e. variables can change type in some circumstances. Examples of what happens when variables are explicitly converted using global type conversion functions are set out below. It is important to bear in mind that type conversion can also happen implicitly, e.g. if we try to concatenate a number with a string then the number will be converted to a string beforehand.

Original value (x)	Result of conversion to a string, using String(x)	Result of conversion to a number, using Number(x)	Result of conversion to a Boolean, using Boolean(x)
false	"false"	0	false
true	"true"	1	True
0	"0"	0	false
1	"1"	1	True
"0"	"0"	0	false
"1"	"1"	1	True
NaN	"NaN"	NaN	false
Infinity	"Infinity"	Infinity	True
-Infinity	"-Infinity"	-Infinity	True
""	""	0	false
"10"	"10"	10	True
"ten"	"ten"	NaN	True
[ ]	""	0	True
[10]	"10"	10	True

[5,10]	"5,10"	NaN	True
["ten"]	"ten"	NaN	True
["five","ten"]	"five","ten"	NaN	True
function(){}	"function(){}"	NaN	True
{ }	"[object Object]"	NaN	True
null	"null"	0	False
undefined	"undefined"	NaN	False

It is worth noting that some of the global type conversion functions are mimicked by type conversion functions attached to specific object types. The behaviours of the two apparently similar functions are not always identical, as the ones attached to specific object types will include a conversion into the relevant type before there is an attempt to convert the variable into its eventual output type.

Recent developments:

In recent years, the capabilities and therefore sophistication of the JavaScript language has grown considerably. Browser developers have put a considerable amount of effort into arranging for JavaScript code to run as quickly as possible within browsers. For example, they have developed 'just in time' compilation techniques to supplant older purely interpreted ways of executing the JavaScript statements. JavaScript as a language has been standardised and object-orientated programming tools such as error trapping have been added. Event handling (which is particularly important for browser based programs) has been further refined. Its DOM has continued to evolve and become more sophisticated as website usage has expanded and evolved.

Appendix A: HTML Elements

[HTMLElements]

Conventionally in HTML, everything between the start of a start tag and the end of its corresponding end tag is called an *element*. The element content is the material between the start and end tag. In HTML, some tags are automatically empty, such as `<br>` and hence don't have any element content or end tag. For example:

HTML Element	Start tag	Element content	End tag
`<h1>My heading</h1>`	`<h1>`	`My heading`	`</h1>`
`<p>My paragraph</p>`	`<p>`	`My paragraph`	`</p`
` `	` `		

The following is a list of HTML elements / tags. HTML 5 is the latest available version of HTML as at the time of writing. Some elements allowable in previous HTML versions have been discontinued in it and it is advisable not to use these elements anymore. Conventions used to open and close elements in XHTML are not quite the same as those used in HTML.

Tag	Description	More	Further comments
`<!-- … -->`	A (potentially multiline) comment	Here	
`<!DOCTYPE>`	Document type	Here	
`<a>`	Hyperlink	Here	
`<abbr>`	Abbreviation or acronym	Here	
`<acronym>`	Acronym	Here	Not supported in HTML 5
`<address>`	Contact information for the author or owner of a document	Here	
`<applet>`	Embedded applet	Here	Not supported in HTML 5 (instead use <embed> or <object>)
`<area>`	An area inside an image map	Here	
`<article>`	Article	Here	New in HTML 5
`<aside>`	Content aside from the page content	Here	New in HTML 5
`<audio>`	Sound content	Here	New in HTML 5
`<b>`	Bold text	Here	
`<base>`	The base URL/target for all relative URLs in a document	Here	
`<basefont>`	Default font, colour and size of all text in a document	Here	Not supported in HTML 5 (instead use CSS)
`<bdi>`	Isolates a part of text that might be formatted in a different direction to other text outside that part	Here	New in HTML 5
`<bdo>`	Overrides the current text direction	Here	
`<big>`	Big text	Here	Not supported in HTML 5 (instead use CSS)
`<blockquote>`	Section that is quoted from another source	Here	
`<body>`	The body of the document	Here	
` `	A single line break (c.f. a carriage return)	Here	
`<button>`	A clickable button	Here	

<canvas>	Used to draw graphics via scripting (usually via JavaScript)	Here	New in HTML 5
<caption>	Table caption	Here	
	Centred text	Here	Not supported in HTML 5 (instead use CSS)
<cite>	Title of a work	Here	
<code>	A piece of computer code	Here	
<col>	Indicates column properties assigned to each column within a <colgroup> element	Here	
<colgroup>	Specifies a group of one or more columns in a table	Here	
<data>	Links content with a machine-readable translation	Here	New in HTML 5
<datalist>	A list of pre-defined options for an <input> control	Here	New in HTML 5
<dd>	Description or value of an entry in a description list	Here	
<del>	Indicates text deleted from a document	Here	
<details>	Additional details that a user can view or hide	Here	New in HTML 5
<dfn>	Defining instance of a term	Here	
<dialog>	Dialog box or window	Here	
<dir>	Directory list	Here	Not supported in HTML 5 (instead use)
	Section in a document	Here	Usually assigned its own style
<dl>	Description list	Here	
<dt>	Term or name in a description list	Here	
	Emphasised text	Here	Often used to italicise text, but ideally this should be done using CSS
<embed>	A container for an external (non-HTML) application	Here	New in HTML 5
<fieldset>	Groups related elements in a form	Here	
<figcaption>	A caption for a <figure> element	Here	New in HTML 5
<figure>	Self-contained content	Here	New in HTML 5
<font>	Font, colour and size of text	Here	Not supported in HTML 5 (instead use CSS)
<footer>	Footer for a document or section	Here	New in HTML 5
<form>	An HTML form for user input	Here	
<frame>	A window (frame) within a frameset	Here	Not supported in HTML 5
<frameset>	A set of <frame> elements	Here	Not supported in HTML 5
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	HTML headings	Here	Provides a hierarchy of headings
<head>	Provides information about the document	Here	
<header>	Header for a document or section	Here	New in HTML 5

<hr>	Indicates a thematic change in the content	Here	Often rendered as a line across the window
<html>	Is the root node of an HTML document	Here	Only <!DOCTYPE> elements should appear outside the <html> element
<i>	A part of text in an alternate voice or mood	Here	
<iframe>	An inline frame	Here	
<img>	An image	Here	
<input>	A (single-line) input control	Here	
<ins>	Indicates text added to a document	Here	
<kbd>	Keyboard input	Here	
<keygen>	A key-pair generator field (for forms)	Here	New in HTML 5
<label>	A label for an <input> element	Here	
<legend>	Caption for a <fieldset> element	Here	
	A list item	Here	
<link>	Defines the relationship between a document and an external resource	Here	Most commonly used to link to style sheets
<main>	Main content of a document	Here	New in HTML 5
<map>	A client-side image-map	Here	
<mark>	Marked/highlighted text	Here	New in HTML 5
<menu>	Menu or list of commands	Here	
<menuitem>	A menu item/command that a user can invoke from a popup menu	Here	New in HTML 5
<meta>	Metadata about an HTML document	Here	
<meter>	A scalar measurement within a specific range (a gauge)	Here	New in HTML 5
<nav>	Navigation links	Here	New in HTML 5
<noframes>	Alternate content for users whose browsers do not support frames	Here	Not supported in HTML 5
<noscript>	Alternate content for users whose browsers do not support client-side scripts	Here	
<object>	Embedded object	Here	
	Ordered list	Here	
<optgroup>	Group of related options in a drop-down list	Here	
<option>	An option in a drop-down list	Here	
<output>	The result of a calculation	Here	New in HTML 5
<p>	Paragraph	Here	
<param>	Parameter for an object	Here	
<picture>	A container for multiple image resources	Here	New in HTML 5
<pre>	Preformatted text	Here	
<progress>	Represents the progress of a task	Here	New in HTML 5
<q>	Short quotation	Here	
<rp>	Indicates what to show in browsers that do not support ruby annotations	Here	New in HTML 5
<rt>	Explanation / pronunciation of	Here	New in HTML 5. For East

	characters		Asian typography
<ruby>	Ruby annotation	Here	New in HTML 5. For East Asian typography
<s>	Text that is no longer correct	Here	
<samp>	Sample output from a computer program	Here	
<script>	Client-side script	Here	Usually written in JavaScript
<section>	Section in a document	Here	New in HTML 5
<select>	A drop-down list	Here	
<small>	Smaller text	Here	
<source>	Allows multiple media resources for media elements	Here	New in HTML 5. Links together associated <video> and <audio>
	Section in a document	Here	Usually defined with its own style
<strike>	Strikethrough text	Here	Not supported in HTML 5 (instead use <del> or <s>)
	Defines more important text	Here	Commonly used as a way of highlighting text or making it bold
<style>	Style information for a document	Here	
<sub>	Subscripted text	Here	
<summary>	Heading for a <details> element	Here	New in HTML 5
<sup>	Superscripted text	Here	
<table>	Table	Here	
<tbody>	Body of a table	Here	
<td>	Table cell (within a table row)	Here	
<textarea>	Multiline input control	Here	
<tfoot>	Footer content of a table	Here	
<th>	Table header cell	Here	
<thead>	Header content of a table	Here	
<time>	Date / time	Here	New in HTML 5
<title>	Title for document	Here	Appears in <head>
<tr>	Table row (within a table)	Here	
<track>	Text track for a media element	Here	New in HTML 5 for <video> and <audio>
<tt>	Teletype text	Here	Not supported in HTML 5 (instead use CSS)
<u>	Text that should be stylistically different from normal text	Here	Commonly used for underlining
	Unordered list	Here	
<var>	Variable	Here	
<video>	Video or movie	Here	New in HTML 5
<wbr>	Possible line-break	Here	New in HTML 5

In practice, we can group HTML elements into a smaller number of categories:

(a) Basic elements (and tags that don't contain anything)
(b) Audio / video

(c) Formatting
(d) Forms and inputs
(e) Frames
(f) Images
(g) Links
(h) Lists
(i) Metadata
(j) Programming (i.e. scripting)
(k) Tables
(l) Styles (and other miscellaneous elements)

Some of the formatting elements are called phrase elements, as they are typically used primarily to delineate specific types of text.

Basic elements (and elements that don't contain anything)

[HTMLElementsBasic]

The following is a list of HTML basic elements that every HTML page is supposed to contain (although if a `<!DOCTYPE>` element is not present then essentially all modern browsers will assume that the page is an HTML page, and, as explained in HTML Getting Started, you can often also dispense with the `<title>` element (and potentially also the `<html>`, `<head>` and `<body>` elements).

Tag	Description	More	Further comments
`<!DOCTYPE>`	Document type	Here	
`<body>`	The body of the document	Here	
`<head>`	Provides information about the document	Here	
`<html>`	Is the root node of an HTML document	Here	Only <!DOCTYPE> elements should appear outside the <html> element
`<title>`	Title for document	Here	Appears in <head>

The `<head>` element can also be deemed a metadata element, as it forms part of the way in which metadata is included in such a document.

Three other elements are also typically considered 'basic', either because they don't contain anything or because they comment out other material:

Tag	Description	More	Further comments
`<!-- … -->`	A (potentially multiline) comment	Here	
` `	A single line break (c.f. a carriage return)	Here	
`<hr>`	Indicates a thematic change in the content	Here	Often rendered as a line across the window

The default styles applicable to these elements are shown here.

Audio / video elements

[HTMLElementsAudioVideo]

The following is a list of HTML audio / video elements:

Tag	Description	More	Further comments
<audio>	Sound content	Here	New in HTML 5
<source>	Allows multiple media resources for media elements	Here	Links together associated <video> and <audio>
<video>	Video or movie	Here	New in HTML 5

The default styles applicable to these elements are shown here.

Formatting elements

[HTMLElementsFormatting]

The following is a list of HTML formatting elements:

Tag	Description	More	Further comments
<abbr>	Abbreviation or acronym	Here	
<acronym>	Acronym	Here	Not supported in HTML 5
<address>	Contact information for the author or owner of a document	Here	
<b>	Bold text	Here	
<basefont>	Default font, colour and size of all text in a document	Here	Not supported in HTML 5 (instead use CSS)
<bdi>	Isolates a part of text that might be formatted in a different direction to other text outside that part	Here	New in HTML 5
<bdo>	Overrides the current text direction	Here	
<big>	Big text	Here	Not supported in HTML 5 (instead use CSS)
<blockquote>	Section that is quoted from another source	Here	
	Centred text	Here	Not supported in HTML 5 (instead use CSS)
<cite>	Title of a work	Here	
<code>	A piece of computer code	Here	
<del>	Indicates text deleted from a document	Here	
<dfn>	Defining instance of a term	Here	
	Emphasised text	Here	Often used to italicise text, but ideally this should be done using CSS
<font>	Font, colour and size of text	Here	Not supported in HTML 5 (instead use CSS)
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>	HTML headings	Here	Provides a hierarchy of headings

<i>	A part of text in an alternate voice or mood	Here	
<ins>	Indicates text added to a document	Here	
<kbd>	Keyboard input	Here	
<mark>	Marked/highlighted text	Here	New in HTML 5
<meter>	A scalar measurement within a specific range (a gauge)	Here	New in HTML 5
<p>	Paragraph	Here	
<pre>	Preformatted text	Here	
<progress>	Represents the progress of a task	Here	New in HTML 5
<q>	Short quotation	Here	
<rp>	Indicates what to show in browsers that do not support ruby annotations	Here	New in HTML 5
<rt>	Explanation / pronunciation of characters	Here	New in HTML 5. For East Asian typography
<ruby>	Ruby annotation	Here	New in HTML 5. For East Asian typography
<s>	Text that is no longer correct	Here	
<samp>	Sample output from a computer program	Here	
<small>	Smaller text	Here	
<strike>	Strikethrough text	Here	Not supported in HTML 5 (instead use <del> or <s>)
	Defines more important text	Here	Commonly used as another way of highlighting text or making it bold
<sub>	Subscripted text	Here	
<sup>	Superscripted text	Here	
<time>	Date / time	Here	New in HTML 5
<tt>	Teletype text	Here	Not supported in HTML 5 (instead use CSS)
<u>	Text that should be stylistically different from normal text	Here	Commonly used for underlining
<var>	Variable	Here	
<wbr>	Posible line-break	Here	New in HTML 5

The default styles applicable to these elements are shown here. The behaviour of most formatting elements can be replicated using CSS.

Forms and inputs elements

[HTMLElementsFormsAndInputs]

The following is a list of HTML forms and inputs elements:

Tag	Description	More	Further comments
<datalist>	A list of pre-defined options for input controls	Here	New in HTML 5
<fieldset>	Groups related elements in a form	Here	

<form>	An HTML form for user input	Here	
<input>	A (single-line) input control	Here	
<keygen>	A key-pair generator field (for forms)	Here	New in HTML 5
<label>	A label for an <input> element	Here	
<legend>	Caption for a <fieldset> element	Here	
<optgroup>	Group of related options in a drop-down list	Here	
<option>	An option in a drop-down list	Here	
<output>	The result of a calculation	Here	New in HTML 5
<select>	A drop-down list	Here	
<textarea>	Multiline input control	Here	

The default styles applicable to these elements are shown here.

Frame elements

[HTMLElementsFrames]

The following is a list of HTML frame elements:

Tag	Description	More	Further comments
<frame>	A window (frame) within a frameset	Here	Not supported in HTML 5
<frameset>	A set of <frame> elements	Here	Not supported in HTML 5
<iframe>	An inline frame	Here	
<noframes>	Alternate content for users whose browsers do not support frames	Here	Not supported in HTML 5

The default styles applicable to these elements are shown here.

Image elements

[HTMLElementsImages]

The following is a list of HTML image elements:

Tag	Description	More	Further comments
<area>	An area inside an image map	Here	
<canvas>	Used to draw graphics via scripting (usually via JavaScript)	Here	New in HTML 5
<figcaption>	A caption for a <figure> element	Here	New in HTML 5
<figure>	Self-contained content	Here	New in HTML 5
<img>	An image	Here	
<map>	A client-side image-map	Here	
<picture>	A container for multiple image resources	Here	New in HTML 5

The default styles applicable to these elements are shown here.

Link elements

[HTMLElementsLinks]

The following is a list of HTML link elements:

Tag	Description	More	Further comments
<a>	Hyperlink	Here	
<link>	Defines the relationship between a document and an external resource	Here	Most commonly used to link to style sheets
<nav>	Navigation links	Here	New in HTML 5

The default styles applicable to these elements are shown here.

List elements

[HTMLElementsLists]

The following is a list of HTML list elements:

Tag	Description	More	Further comments
<dd>	Description or value of an entry in a description list	Here	
<dir>	Directory list	Here	Not supported in HTML 5 (instead use)
<dl>	Description list	Here	
<dt>	Term or name in a description list	Here	
	A list item	Here	
<menu>	Menu or list of commands	Here	
<menuitem>	A menu item/command that a user can invoke from a popup menu	Here	New in HTML 5
	Ordered list	Here	
	Unordered list	Here	

The default styles applicable to these elements are shown here.

Metadata elements

[HTMLElementsMetadata]

The following is a list of HTML metadata elements:

Tag	Description	More	Further comments
<base>	The base URL/target for all relative URLs in a document	Here	
<head>	Provides information about the document	Here	
<meta>	Metadata about an HTML document	Here	

The default styles applicable to these elements are shown here.

Programming elements

[HTMLElementsProgramming]

The following is a list of HTML programming elements:

Tag	Description	More	Further comments
<applet>	Embedded applet	Here	Not supported in HTML 5 (instead use <embed> or <object>)
<embed>	A container for an external (non-HTML) application	Here	New in HTML 5
<noscript>	Alternate content for users whose browsers do not client-side scripts	Here	
<object>	Embedded object	Here	
<param>	Parameter for an object	Here	
<script>	Client-side script	Here	Usually written in JavaScript

The default styles applicable to these elements are shown here.

Table elements

[HTMLElementsTables]

The following is a list of HTML table elements:

Tag	Description	More	Further comments
<caption>	Table caption	Here	
<col>	Indicates column properties assigned to each column within a <colgroup> element	Here	
<colgroup>	Specifies a group of one or more columns in a table	Here	
<table>	Table	Here	
<tbody>	Body of a table	Here	
<td>	Table cell (within a Table row)	Here	
<tfoot>	Footer content in a table	Here	
<th>	Table header cell	Here	
<thead>	Header content in a table	Here	
<tr>	Table row (within a Table)	Here	

The default styles applicable to these elements are shown here.

Style (and other miscellaneous) elements

[HTMLElementsStyles]

The following is a list of HTML style (and other miscellaneous) elements:

Tag	Description	More	Further comments

<article>	Article	Here	New in HTML 5
<aside>	Content aside from the page content	Here	New in HTML 5
<data>	Links content with a machine-readable translation	Here	New in HTML 5
<details>	Additional details that a user can view or hide	Here	New in HTML 5
<dialog>	Dialog box or window	Here	
	Section in a document	Here	Usually assigned its own style
<footer>	Footer for a document or section	Here	New in HTML 5
<header>	Header for a document or section	Here	New in HTML 5
<main>	Main content of a document	Here	New in HTML 5
<section>	Section in a document	Here	New in HTML 5
	Section in a document	Here	Usually defined its own style
<style>	Style information for a document	Here	
<summary>	Heading for a <details> element	Here	New in HTML 5

The default styles applicable to these elements are shown here.

Phrase elements

[HTMLPhraseElements]

Some HTML formatting elements are typically used to delineate text of specific types. These HTML elements are called 'phrase' elements:

Tag	Description	More	Further comments
<code>	A piece of computer code	Here	
	Emphasised text	Here	Often used to italicise text, but ideally this should be done using CSS
<kbd>	Keyboard input	Here	
<samp>	Sample output from a computer program	Here	
	Defines more important text	Here	Commonly used as a way of highlighting text or making it bold
<var>	Variable	Here	

The default styles applicable to these elements are shown here.

XHTML

[HTMLTutorialXHTML]

XHTML stands for eXtensible Hypertext Markup Langauge. It is designed to be very like HTML but structured in a fashion that also adheres to the rules of XML.

Typically, most browsers accept some types of 'badly formed' HTML, e.g. HTML in which a document's <head> element is not properly closed before its <body> element is opened. This is despite such markup text failing to adhere to the rules that HTML is supposed to follow. However, such pages may not work well or consistently on some devices. A processing overhead is incurred when a browser tries to interpret badly-formed HTML, which may not be practical for some smaller devices. There may also be several possible ways of interpreting badly-formed HTML. XML is more rigidly structured than HTML (and it is easier to test that its rules are being adhered to), making it an easier vehicle through which to introduce disciplines that aim to ensure all markup text is 'well-formed'.

The main differences between HTML and XHTML are:

1. The XHTML DOCTYPE element (which takes the form `<!DOCTYPE` *attributes*`>`) must be present at the start of the document.
2. <html>, <head>, <title> and <body> elements must also be present, and the `xmlns` attribute of the <html> element must be defined appropriately.
3. All XHTML elements must be properly closed (and properly nested), e.g. using `</p>` to close a paragraph (<p>) element and not just starting a new one with a new `<p>`. Note, usually browsers would interpret `<p>` *text1* `<p>` *text2* `</p>` as two consecutive paragraphs even though this involves badly-formed HTML.
4. A corollary of 3. is that HTML empty elements such as the <br>, <hr> and <img> element must also be properly closed in XHTML, i.e. they should be written as `<br />`, `<hr />` and `<img src="`*filename*`" />` respectively.
5. XHTML element and attribute names must use lower case, e.g. the XHTML <p> element must be written as `<p>` *text* `</p>` rather than `<P>` *text* `</P>`.
6. All XHTML attribute values must be included in quotes. So, HTML such as `<p width=100px>` is wrong in XHTML and should be replaced by `<p width="100px">`.
7. Attribute minimisation is forbidden. Attribute minimisation in HTML involves including just the attribute name rather than both the name and its value if its value is the same as its name. For example, HTML of the form `<input type="checkbox" name="fruit" value="apple" checked />` should be replaced in HTML by `<input type="checkbox" name="fruit" value="apple" checked="checked" />`.

In practice, it is usually quite easy (if possibly laborious) to convert HTML to XHTML by:

(a) Adding a suitable XHTML `<!DOCTYPE>` statement to the first line of the page and adding an `xmlns` attribute to the html element of the page
(b) Changing all element names and attribute names to lower case
(c) Closing all empty elements
(d) Putting all attribute values in quotes (and eliminating any attribute minimisation that is present)

An example of a minimal XHTML page is:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <title>Title</title>
</head>
<body>
    Content
</body>
</html>
```

Individual HTML Elements:

<a>

[HTMLElementA]

The HTML <a> or anchor element represents a hyperlink. It typically takes the form:

```
<a href="url">text</a>
```

or

```
<a href="url" target="x">text</a>
```

Here:

- *text* is what the user sees
- the href attribute contains the destination address of the link (i.e. where the browser goes to when the hyperlink is clicked). This could be a web address, e.g. http://www.nematrian.com/Introduction.aspx, a local link (to the same website as the page, e.g. `introduction.aspx`) or a bookmark within a resource
- the target attribute indicates where to open the linked document

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`download`	Target will be downloaded when user clicks on hyperlink	Here
`href`	URL of page the link goes to	Here
`hreflang`	Language of linked document	Here
`media`	Specifies media / device linked document is optimised for	Here
`rel`	Relationship between current document and linked document	Here
`target`	Specifies where / how to open the linked document (or where to submit the form)	Here
`type`	Type of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (except perhaps the `media` attribute). It also has a `text` property which sets or returns the text content of the object. It also supports the `hash`, `host`, `hostname`, `origin`, `password`, `pathname`, `port`, `protocol`, `search` and `username` variants of the `href` attribute, see here for more details.

By default, an unvisited link is underlined and is usually blue, a visited link is underlined and purple and an active link is underlined and is usually purple, see here. However, it is possible to change these defaults by setting relevant CSS attributes.

<abbr>

[HTMLElementAbbr]

The HTML `<abbr>` element indicates an abbreviation or acronym. The full text which is being abbreviated can be included in the element's title attribute and it will then typically appear as tooltip text if the mouse is moved over the element.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<acronym>

[HTMLElementAcronym]

The HTML `<acronym>` element was used to indicate an acronym. It is not supported in HTML 5. Instead, use the <del> or <s> element.

<address>

[HTMLElementAddress]

The HTML `<address>` element is usually used to define contact information for the author or owner of a document or file. If it is inside an <article> element then it typically represents contact information for that article. If it is outside an article element but inside a <body> element then it typically represents contact information for the document or page.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<applet>

[HTMLElementApplet]

The HTML `<applet>` element was used to indicate an embedded applet. It is not supported in HTML 5. Instead, use the <embed> or <object> element.

<area>

[HTMLElementArea]

The HTML `<area>` element identifies an area inside an image-map.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`alt`	Specifies alternative text to show when original content fails to display	Here
`coords`	Specifies the coordinates of an <area>	Here
`download`	Target will be downloaded when user clicks on hyperlink	Here
`href`	URL of page the link goes to	Here
`hreflang`	Language of linked document	Here
`media`	Specifies media / device linked document is optimised for	Here
`rel`	Relationship between current document and linked document	Here
`shape`	Specifies shape of an <area> element	Here
`target`	Specifies where / how to open the linked document (or where to submit the form)	Here
`type`	Type of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (except the `download`, `hreflang`, `media`, `rel` and `type` attribute). The corresponding HTML DOM object also typically supports the `hash`, `host`, `hostname`, `origin`, `password`, `pathname`, `port`, `protocol`, `search` and `username` variants of the `href` attribute, see here for more details. The default style applicable to this element is shown here.

<article>

[HTMLElementArticle]

The HTML `<article>` element indicates a self-contained piece of content, such as a blog or forum post, a specific news story or some self-contained comment on a specific piece of text. It is new in HTML 5.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<aside>

[HTMLElementAside]

The HTML `<aside>` element indicates some content separate from but related to surrounding context. It is new in HTML 5.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<audio>

[HTMLElementAudio]

The HTML `<audio>` element is used to define and play sound, such as music or other audio streams. Supported file formats include MP3 (nearly all browsers), Wav (not Internet Explorer) and Ogg (some browsers). It is new in HTML 5.

If the browser does not support `<audio>` elements then any text between the `<audio>` and `</audio>` tags will be displayed.

The attributes it can take (in addition to HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`autoplay`	Specifies whether media should start playing as soon as ready	Here
`controls`	Whether controls (such as play and pause buttons) should be displayed	Here
`loop`	Media to start over again when it finishes	Here
`muted`	Audio output should be muted	Here
`preload`	If / how author thinks media should be loaded when page loads	Here
`src`	URL of media	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports DOM generic media properties and methods and the following additional properties and methods.

Additional methods:

Method	Description	More
`fastSeek()`	Seeks to a specified time in audio	Here
`getStartDate()`	Returns Date object representing current timeline offset	Here

The default style applicable to this element is shown here.

<b>

[HTMLElementB]

The HTML <b> element indicates bold text. According to the HTML 5 specification it should be used as a last resort when no other elements such as , <h1>, <h2>, <h3>, <h4>, <h5> or <h6> are appropriate.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<base>

[HTMLElementBase]

The HTML <base> element specifies the base URL for all relative URLs in a document. There can be at most one <base> element in any specific document (and it should be inside the relevant <head> element).

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
href	URL of page the link goes to	Here
target	Specifies where / how to open the linked document (or where to submit the form)	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also typically supports the hash, host, hostname, origin, password, pathname, port, protocol, search and username variants of the href attribute, see here for more details. The default style applicable to this element is shown here.

<basefont>

[HTMLElementBasefont]

The HTML <basefont> element was used to indicate the default font, colour and size of all text in a document. It is not supported in HTML 5. Instead, use CSS.

<bdi>

[HTMLElementBdi]

The HTML <bdi> element indicates material that might be formatted in a different direction from other material surrounding the element. 'bdi' stands for 'bi-directional isolation'. It is new in HTML 5. A similar effect can usually be achieved using the unicode-bidi style applied to e.g. a span element, but the semantic meaning is only conveyed by the <bdi> element and in some cases browsers may ignore CSS, but not a <bdi> element.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<bdo>

[HTMLElementBdo]

The HTML <bdo> element makes it possible to override the current text direction.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
dir	Text direction for element content	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<big>

[HTMLElementBig]

The HTML <big> element was used to indicate big text. It is not supported in HTML 5. Instead, use CSS.

<blockquote>

[HTMLElementBlockquote]

The HTML <blockquote> element indicates a section that is quoted from another source. Browsers often indent text in such elements (typically a <q> element is used for an in-line quotation and isn't indented).

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
cite	URL which explains the quote / deleted / inserted text	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<body>

[HTMLElementBody]

The HTML <body> element identifies the body of the document and contains all the visible contents of the document, such as text, hyperlinks, tables and images etc.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the alink, background, bgcolor, link, text and vlink attributes, but these are no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

[HTMLElementBr]

The HTML <br> element indicates a single line break (c.f. a carriage return). In XHTML, it needs to be 'properly' closed as per <br />.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<button>

[HTMLElementButton]

The HTML <button> element indicates a clickable button. Inside a <button> element (unlike an <input> element) you can put content such as text or images. You should typically specify the type attribute for <button> element as different browsers do not necessarily default to the same type. Within a <form> element you should also bear in mind that different browsers may submit different values.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
autofocus	Specifies whether element should automatically get focus when page loads	Here
disabled	Specified element(s) to be disabled	Here
form	Name of the form that element belongs to	Here
formaction	Where to send form-data to when form submitted	Here. Only for type = submit
formenctype	How form-data should be encoded before sending it to a server	Here. Only for type = submit
formmethod	How to send form-data (i.e. which HTTP method to use)	Here. Only for type = submit

`formnovalidate`	Specifies that form-data should not be validated on submission	Here. Only for type = `submit`
`formtarget`	Specifies where to display the response that is received after submitting form	Here. Only for type = `submit`
`name`	Name of element	Here
`type`	Type of element	Here
`value`	Value of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. However, the DOM versions of formaction, formenctype, formmethod, formnovalidate and formtarget need to adopt the JavaScript name capitalisation convention, i.e. need to be: `formAction`, `formEnctype`, `formMethod`, `formNoValidate` and `formTarget` respectively. The default style applicable to this element is shown here.

<canvas>

[HTMLElementCanvas]

The HTML `<canvas>` element is used to draw graphics via scripting (usually via JavaScript). It is new in HTML 5. Such an element isn't directly endowed with its own drawing abilities. Instead it is necessary to apply the `getContext()` method to the corresponding DOM object to return another object that does have drawing abilities.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`height`	Height of element	Here
`width`	Width of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, as well as the following additional methods:

Method	Description	More
`getContext()`	Returns an object that can be used to elaborate (populate) the canvas	Here
`restore()`	Returns previously saved path state and attributes	Here
`save()`	Saves path state and attributes	Here

Older versions of getContext focus on `getContext("2d")` which allows drawing of many types of two-dimensional material (e.g. text, lines, boxes, circles etc.). Newer versions support drawing of hardware-supported three-dimensional objects via `getContext("WebGL")` or `getContext("WebGL2")`.

The object returned by the `getContext("2d")` method provides the following properties and methods:

Additional getContext("2d") properties:

Property	Description	More
Styles etc.		
`fillStyle`	Sets / returns style (colour, gradient, pattern etc.) used to fill element	Here
`strokeStyle`	Sets / returns style used for strokes	Here
`shadowBlur`	Sets / returns shadow blur level, see CSS text-shadow property	Here
`shadowColor`	Sets / returns shadow colour, see CSS text-shadow property	Here
`shadowOffsetX`	Sets / returns shadow horizontal offset, see CSS text-shadow property	Here
`shadowOffsetY`	Sets / returns shadow vertical offset, see CSS text-shadow property	Here
Line styles		
`lineCap`	Sets / returns style used for line ends	Here
`lineJoin`	Sets / returns type of corner between two lines where they join	Here
`lineWidth`	Sets / returns line width	Here
`miterLimit`	Sets / returns maximum mitre limit	Here
Text		
`font`	Sets / returns CSS font property for current text	Here
`textAlign`	Sets / returns CSS text-align property for current text	Here
`textBaseline`	Sets / returns text baseline for current text	Here
Sizing and manipulation of individual pixels		
`data`	Returns object containing image data for specific ImageData object	Here
`height`	Returns height of an ImageData object	Here
`width`	Returns width of an ImageData object	Here
Other		
`globalAlpha`	Sets / returns current alpha, i.e. transparency value, of drawing	Here
`globalCompositeOperation`	Sets / returns how new images are drawn onto existing images	Here

Additional getContext("2d") methods:

Method	Description	More
Styles etc.		
`addColorStop()`	Specifies colours and stop positions for a gradient object	Here
`createLinearGradient()`	Creates a linear gradient	Here
`createPattern()`	Repeats a specific element in a specific direction	Here
`createRadialGradient()`	Creates a radial (i.e. circular) gradient	Here
Rectangles		
`clearRect()`	Clears specified pixels within a rectangle	Here
`fillRect()`	Draws a 'filled' rectangle	Here

`rect()`	Creates a rectangle	Here
`strokeRect()`	Draws a rectangle that is not 'filled'	Here
Paths		
`arc()`	Creates a circular arc	Here
`arcTo()`	Creates a circular arc between two tangents	Here
`beginPath()`	Begins / resets a path	Here
`bezierCurveTo()`	Creates cubic Bézier curve	Here
`clip()`	Clips region from canvas	Here
`closePath()`	Completes path back to its original starting point	Here
`fill()`	Fills current path	Here
`isPointInPath()`	Returns true if point is in current path, otherwise false	Here
`lineTo()`	Moves path to a specified point in the canvas, creating a line from the previous point	Here
`moveTo()`	Moves path to a specified point in the canvas without creating a line	Here
`quadraticCurveTo()`	Creates quadratic Bézier curve	Here
`stroke()`	Draws path in the canvas	Here
Transformations		
`rotate()`	Rotates current drawing	Here
`scale()`	Scales current drawing	Here
`setTransform()`	Defines a transform matrix and then applies `transform()` method	Here
`transform()`	Applies a transformation to current drawing	Here
`translate()`	Applies a translation to current drawing (i.e. adjusts the position of its origin)	Here
Text		
`fillText()`	Draws 'filled' text	Here
`measureText()`	Returns object indicating width of specified text	Here
`strokeText()`	Draws text that is not 'filled'	Here
Drawing images		
`drawImage()`	Draws an image, canvas or video onto canvas	Here
Sizing and manipulation of individual pixels		
`createImageData()`	Creates a new blank ImageData object	Here
`getImageData()`	Returns ImageData object characterised by pixel data for specific rectangle in canvas	Here
`putImageData()`	Puts image data included in an ImageData object onto canvas	Here

The objects returned by the `getContext("WebGL")` and the `getContext("WebGL2")` methods provide equivalent 3D and other more sophisticated graphical capabilities. They are not currently explored further in these pages.

The default style applicable to this element is shown here.

<caption>

[HTMLElementCaption]

The HTML <caption> element indicates a table caption. It should be inserted immediately after opening <table> tag of the <table> element.

The attributes it can take are HTML global attributes and HTML event attributes. It used to support the align attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

[HTMLElementCenter]

The HTML element was used to indicate centred text. It is not supported in HTML 5. Instead, use CSS.

<cite>

[HTMLElementCite]

The HTML <cite> element indicates the title of a work (e.g. a book or movie). It is not typically used for the author of the work.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<code>

[HTMLElementCode]

The HTML <code> element is a phrase element indicating a piece of computer code. It is not depreciated, but typically a richer effect can be achieved using CSS.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<col>

[HTMLElementCol]

The HTML <col> element indicates the column properties assigned to each column within a <colgroup> element. It can be used to apply styles to entire columns in a table.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`span`	Number of columns to span	Here

It also used to support the `align`, `char`, `charoff`, `valign` and `width` attributes, but these are no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<colgroup>

[HTMLElementColgroup]

The HTML `<colgroup>` element specifies a group of one or more columns in a table. It can be used to apply styles to entire columns in a table. It must be a child of a <table> element, positioned after any <caption> elements and before any <tbody>, <tfoot>, <thead> and <tr> elements. If you want to define different styles to column within the column group then use the <col> element.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`span`	Number of columns to span	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

comment

[HTMLElementComment]

The HTML *comment* element takes the form:

```
<!-- … -->
```

It can be potentially multiline. All text within the comment is ignored.

HTML comment elements do not in effect support any meaningful attributes.

<data>

[HTMLElementData]

The HTML `<data>` element links content with a machine-readable translation. It is new in HTML 5. If the content is date or time based then an alternative is to use a <time> element.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
value	Value of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<datalist>

[HTMLElementDatalist]

The HTML <datalist> element identifies a list of pre-defined options for an <input> element. It is new in HTML 5. The <input> element's list attribute should be set to the id of the <datalist> in order to bind the two together and within the <datalist> should be some <option> elements. Users will then see a dropdown list of choices (defined by the <option> elements) that they can select for the <input> element, e.g.:

```
<input list="fruit">
<datalist id="fruit">
    <option value="apple">
    <option value="banana">
    <option value="orange">
</datalist>
```

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. It also supports the following additional property:

Property	Description	More
options	Returns a collection of all options included in datalist element	Here

The default style applicable to this element is shown here.

<dd>

[HTMLElementDd]

The HTML <dd> element indicates a description or value of an entry in a description list, i.e. a <dl> element, typically with each <dd> element linked to an associated <dt> element, that defines the term which the <dd> element describes, e.g.:

```
<dl>
    <dt>UK</dt><dd>United Kingdom</dd>
    <dt>USA</dt><dd>United States of America</dd>
</dl>
```

Text, paragraphs, images etc. can be placed inside <dd> and <dt> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<del>

[HTMLElementDel]

The HTML <del> element indicates text deleted from a document. It is often used in association with the <ins> element to highlight modifications to text.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
cite	URL which explains the quote / deleted / inserted text	Here
datetime	Date and time of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the datetime property of the underlying element corresponding to the dateTime property of the DOM object). The default style applicable to this element is shown here.

<details>

[HTMLElementDetails]

The HTML <details> element indicates additional details that a user can view or hide. It is new in HTML 5.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
open	Whether details should be visible (i.e. open) to user	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<dfn>

[HTMLElementDfn]

The HTML <dfn> element indicates the defining instance of a term, usually its first use. In research papers, books and other documents such instances may be italicised. The nearest parent of a <dfn> element should typically contain a short definition or explanation for the term included in the <dfn> element.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<dialog>

[HTMLElementDialog]

The HTML <dialog> element indicates a dialog box or window. It simplifies the creation of popup dialogs, but is not currently supported by all major browsers.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
open	Whether details should be visible (i.e. open) to user	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<dir>

[HTMLElementDir]

The HTML <dir> element was used to indicate a directory list. It is not supported in HTML 5. Instead, use the <ul> element or CSS style lists.

[HTMLElementDiv]

The HTML element indicates a section (i.e. division) in a document. It is usually assigned its own style, differentiating it from other elements next to it in the document.

The attributes it can take are HTML global attributes and HTML event attributes. It used to support the align attribute but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<dl>

[HTMLElementDl]

The HTML <dl> element indicates a description list. It is used in conjunction with <dd> and <dt> elements. A <dt> element identifies a term and the associated <dd> element provides the description for that term, e.g.:

```
<dl>
    <dt>UK</dt><dd>United Kingdom</dd>
    <dt>USA</dt><dd>United States of America</dd>
</dl>
```

Text, paragraphs, images etc. can be placed inside <dd> and <dt> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

document type

[HTMLElementDocumentType]

The HTML *document type* element takes the form:

```
<!DOCTYPE>
```

Or:

<!DOCTYPE *attributes*>

Common declarations include:

Declaration	Meaning
<!DOCTYPE html>	HTML 5
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	HTML 4.01 Strict (excludes depreciated elements)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	HTML 4.01 Transitional (includes depreciated elements but not framesets)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">	HTML 4.01 Frameset (as per Transitional, but also allows use of framesets)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	XHTML 1.0 Strict (excludes depreciated elements)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">	XHTML 1.0 Transitional (includes depreciated elements but not framesets)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">	XHTML 1.0 Frameset (as per Transitional, but also allows

	use of framesets)
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">	XHTML 1.1, as XHTML 1.0 but allows additional modules, e.g. to provide ruby support for East Asian languages

<dt>

[HTMLElementDt]

The HTML <dt> element indicates a term or name in a description list, i.e. a <dl> element, typically with each <dd> element linked to an associated <dt> element, that defines the term which the <dd> element describes, e.g.:

```
<dl>
    <dt>UK</dt><dd>United Kingdom</dd>
    <dt>USA</dt><dd>United States of America</dd>
</dl>
```

Text, paragraphs, images etc. can be placed inside <dd> and <dt> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

[HTMLElementEm]

The HTML element is a phrase element indicating emphasised text. Often it is used to italicise text, but ideally this should be done using CSS.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<embed>

[HTMLElementEmbed]

The HTML <embed> element indicates a container for an external (non-HTML) application, e.g. a plug-in. It is new in HTML 5.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More

height	Height of element	Here
src	URL of resource	Here
type	Type of element	Here
width	Width of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<fieldset>

[HTMLElementFieldset]

The HTML <fieldset> element groups related elements in a form, and typically draws a box around them.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
disabled	Specified element(s) to be disabled	Here
form	Name of the form that element belongs to	Here
name	Name of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above.

It also supports the following additional property:

Property	Description	More
type	Returns the type of the form element that the fieldset element belongs to	Here

The default style applicable to this element is shown here.

<figcaption>

[HTMLElementFigcaption]

The HTML <figcaption> element indicates a caption for a <figure> element. It is new in HTML 5. It can be the first or the last child element of the <figure> element.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<figure>

[HTMLElementFigure]

The HTML <figure> element indicates a piece of self-contained content, like an illustration, diagram or piece of computer code. It is new in HTML 5. Ideally, the content of a <figure> element should not specifically link to its exact position within the text (e.g. in a research paper figures will be referred to in the text, but can be positioned in a variety of places without altering the meaning of the text). A <figcaption> element is used to add a caption to a <figure> element

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<font>

[HTMLElementFont]

The HTML <font> element was used to indicate the font, colour and size of text. It is not supported in HTML 5. Instead, use CSS.

<footer>

[HTMLElementFooter]

The HTML <footer> element indicates a footer for a document or section. It is new in HTML 5. Typically, a <footer> element might contain authorship or copyright information. A document can contain several <footer> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<form>

[HTMLElementForm]

The HTML <form> element indicates an HTML form for user input. Typically, a <form> element will contain one or more of the following form elements:

- <button>
- <fieldset>
- <input>
- <label>
- <optgroup>
- <option>
- <select>
- <textarea>

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
accept-charset	Specifies character encodings used for form submission	Here
action	Where to send form-data when form submitted	Here
autocomplete	Specifies whether element has autocomplete enabled	Here
enctype	How form-data to be encoded when submitted to server	Here. Only for method = post
method	Specifies HTTP method used when sending form-data	Here
name	Name of element	Here
novalidate	Form should not be validated when submitted	Here
target	Specifies where / how to open the linked document (or where to submit the form)	Here

It also used to take the accept attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the novalidate property of the underlying element corresponding to the noValidate property of the DOM object). It also supports the following additional properties and methods:

Additional properties:

Property	Description	More
encoding	An alias for enctype	Here
length	Returns number of elements in form	Here

Additional properties:

Method	Description	More
reset()	Resets form	Here
submit()	Submits form	Here

The default style applicable to this element is shown here.

<frame>

[HTMLElementFrame]

The HTML <frame> element was used to indicate a window (frame) within a frameset. It is not supported in HTML 5.

<frameset>

[HTMLElementFrameset]

The HTML <frameset> element was used to indicate a set of <frame> elements. It is not supported in HTML 5.

<h1>

[HTMLElementH1]

The HTML `<h1>` element indicates a level 1 HTML heading.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<h2>

[HTMLElementH2]

The HTML `<h2>` element indicates a level 2 HTML heading.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<h3>

[HTMLElementH3]

The HTML `<h3>` element indicates a level 3 HTML heading.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<h4>

[HTMLElementH4]

The HTML `<h4>` element indicates a level 4 HTML heading.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<h5>

[HTMLElementH5]

The HTML `<h5>` element indicates a level 5 HTML heading.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<h6>

[HTMLElementH6]

The HTML `<h6>` element indicates a level 6 HTML heading.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<head>

[HTMLElementHead]

The HTML `<head>` element provides information about the document. The `<head>` element can contain the following sorts of elements (it is always supposed to include a <title> element, but HTML that does not do so may not be rejected by browsers):

- <base>
- <link>
- <meta>
- <noscript>
- <script>
- <style>
- <title>

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `profile` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<header>

[HTMLElementHeader]

The HTML `<header>` element indicates a header for a document or section. It is new in HTML 5. Typically, a `<header>` element might contain introductory content, navigational links, one or more heading elements (i.e. <h1>, <h2>, <h3>, <h4>, <h5> or <h6>), a logo and perhaps also some authorship information. A document can contain several `<header>` elements.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

headings

[HTMLElementHeadings]

The HTML <h1>, <h2>, <h3>, <h4>, <h5> and <h6> elements provide a hierarchy of headings (with the <h1> element by default having the largest size and the <h6> element the smallest size).

<hr>

[HTMLElementHr]

The HTML `<hr>` element indicates a thematic change in the content. Often it is rendered as a line across the window.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align`, `noshade`, `size` and `width` attributes, but these are no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<html>

[HTMLElementHtml]

The HTML `<html>` element is the root node of an HTML document. Only <!DOCTYPE> elements should appear outside it. It tells the browser that the document is an HTML document.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`manifest`	Specifies address of document's cache manifest (for offline browsing)	Here
`xmlns`	Indicates the XML namespace attribute (if the content needs to conform to XHTML); is not an HTML attribute as such and is added by default if needed	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<i>

[HTMLElementI]

The HTML `<i>` element indicates a part of text in an alternate voice or mood. Typically it is rendered as italicised text. Convention recommends using the `<i>` element only when there isn't a more appropriate element, such as <cite>, <dfn>, , <mark> or .

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<iframe>

[HTMLElementIframe]

The HTML `<iframe>` element indicates an inline frame. It can also be used to embed another document within the current HTML document.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`height`	Height of element	Here
`name`	Name of element	Here
`sandbox`	Allows an extra set of restrictions for the content of an <iframe> element	Here
`src`	URL of resource	Here
`srcdoc`	HTML content of page to show in an <iframe>	Here
`width`	Width of element	Here

It used to support the `align`, `frameborder`, `longdesc`, `marginheight`, `marginwidth` and `scrolling` attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
`contentDocument`	Returns document object generated by the iframe	Here
`contentWindow`	Returns window object generated by the iframe	Here

The default style applicable to this element is shown here.

[HTMLElementImg]

The HTML `<img>` element indicates an image. It technically has two required attributes, namely `src` (the source of the image) and `alt` (the alternative text displayed if the image cannot be found or cannot be displayed by the browser), although the `alt` attribute can typically be dispensed with. Images are not technically inserted into an HTML page but instead are linked to the page. The `<img>` element therefore creates a placeholder that will include the image once the page is rendered (and the image retrieved by the rendering process from its source location).

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`alt`	Specifies alternative text to show when original content fails to display	Here
`crossorigin`	Specifies how element handles cross-origin requests	Here
`height`	Height of element	Here
`ismap`	Image is a server-side image-map	Here
`sizes`	Specifies size of linked resource	Here
`src`	URL of resource	Here
`srcset`	URL of image to use in different situations	Here
`usemap`	Specifies an image as a client-side image-map	Here
`width`	Width of element	Here

It used to support the `align`, `border`, `hspace`, `longDesc` and `vspace` attributes, but these are no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `crossorigin`, `ismap` and `usemap` properties of the underlying element corresponding to the `crossOrigin`, `isMap` and `useMap` properties of the DOM object). It also supports the following additional properties:

Property	Description	More
`complete`	Returns whether the browser has finished loading the image	Here
`naturalHeight`	Returns original height of image	Here

naturalWidth	Returns original width of image	Here

The default style applicable to this element is shown here.

<input>

[HTMLElementInput]

The HTML <input> element indicates a (single-line) input control into which the user can enter data. It is used within a <form> element. There are many different types of <input> element that vary depending on the type attribute of the element, including:

- button, checkbox, color, date, datetime, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week

In HTML markup <input> elements are empty (they only involve attributes). Labels for input elements can be defined using the <label> element.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
accept	Specifies types of file accepted by server	Here. Only for type = file
alt	Specifies alternative text to show when original content fails to display	Here
autocomplete	Specifies whether element has autocomplete enabled	Here
autofocus	Specifies whether element should automatically get focus when page loads	Here
checked	Specifies that the element should be pre-selected	Here. For type= checkbox or type = radio
dirname	Specifies text direction will be submitted	Here
disabled	Specified element(s) to be disabled	Here
form	Name of the form that element belongs to	Here
formaction	Where to send form-data to when form submitted	Here. Only for type = image and type = submit
formenctype	How form-data should be encoded before sending it to a server	Here. Only for type = image and type = submit
formmethod	How to send form-data (i.e. which HTTP method to use)	Here. Only for type = image and type = submit
formnovalidate	Specifies that form-data should not be validated on submission	Here. Only for type = image and type = submit
formtarget	Specifies where to display the response that is received after submitting form	Here. Only for type = image and type = submit
height	Height of element	Here

list	Refers to <datalist> that contains pre-defined options	Here
max	Maximum value	Here
maxlength	Maximum number of characters allowed in an element	Here
min	Minimum value	Here
multiple	Indicates that a user can enter more than one value	Here
name	Name of element	Here
pattern	Regular expression that value of element is checked against	Here
placeholder	Short hint describing expected value of element	Here
readonly	Whether element is read-only	Here
required	Whether the element must be filled out before submitting form	Here
size	Specifies width in characters of element	Here
src	URL of resource	Here
step	Accepted number intervals	Here
type	Type of element	Here
value	Value of element	Here
width	Width of element	Here

Some of these attributes are valid for only some `<input>` element types:

type	**Valid attributes**
all	autofocus (except for hidden type), disabled (except for hidden type), form, name, type, value
button	-
checkbox	checked, required
color	autocomplete, list
date	autocomplete, max, min, readonly, required, step
datetime	autocomplete, list, max, min, readonly, required, step
datetime-local	autocomplete, list, max, min, readonly, required, step
email	autocomplete, list, maxlength, multiple, pattern, placeholder, readonly, required, size
file	accept, multiple, required
hidden	–
image	alt, formAction, formEnctype, formMethod, formNoValidate, formTarget, height, src, width
month	autocomplete, list, max, min, readonly, required, step
number	autocomplete, list, max, min, placeholder, readonly, required, step
password	autocomplete, maxlength, pattern, placeholder, readonly, required, size
radio	checked, required
range	autocomplete, list, max, min, step
reset	–
search	autocomplete, list, maxlength, pattern, placeholder, readonly, required, size
submit	formAction, formEnctype, formMethod, formNoValidate,

	formTarget
text	autocomplete, list, maxlength, pattern, placeholder, readonly, required, size
time	autocomplete, list, max, min, readonly, required, step
url	autocomplete, list, maxlength, pattern, placeholder, readonly, required, size
week	autocomplete, list, max, min, readonly, required, step

It used to accept the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `formaction`, `formenctype`, `formmethod`, `formnovalidate`, `formtarget`, `maxlength` and `readonly` properties of the underlying element corresponding to the `formAction`, `formEnctype`, `formMethod`, `formNoValidate`, `formTarget`, `maxLength` and `readOnly` properties of the DOM object). It also supports the following additional properties and methods:

Additional properties:

Property	Description	Applies to type
defaultChecked	Returns default value of checked attribute	checkbox, radio. See Here
defaultValue	Sets / returns default value	*All*. See Here
files	Returns a FileList object representing file(s) selected by upload button	file. See Here
form	Returns form that contains element	*All*. See Here
indeterminate	Sets / returns value of its indeterminate status	checkbox. See Here

Additional methods:

Method	Description	Applies to type
select()	Selects content of password field	Password. See Here
stepDown()	Decrements value of relevant field by specified amount	datetime, datetime-local, month, number, range, time, week. See Here
stepUp()	Increments value of relevant field by specified amount	datetime, datetime-local, month, number, range, time, week

The default style applicable to this element is shown here.

<ins>

[HTMLElementIns]

The HTML `<ins>` element indicates text added to a document. It is often used in association with the <del> element to highlight modifications to text.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
cite	URL which explains the quote / deleted / inserted text	Here
datetime	Date and time of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `datetime` property of the underlying element corresponding to the `dateTime` property of the DOM object). The default style applicable to this element is shown here.

<kbd>

[HTMLElementKbd]

The HTML `<kbd>` element is a phrase element indicating keyboard input. It is not depreciated, but typically a richer effect can be achieved using CSS.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<keygen>

[HTMLElementKeygen]

The HTML `<keygen>` element indicates a key-pair generator field (for forms). It is positioned within a form element. When the form is submitted, the private key is stored locally and the public key (of the key-pair) is sent to the server.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
autofocus	Specifies whether element should automatically get focus when page loads	Here
challenge	Indicates value of element should be challenged when submitted	Here
disabled	Specified element(s) to be disabled	Here
form	Name of the form that element belongs to	Here
keytype	Specifies security algorithm of key	Here
name	Name of element	Here

Note: it appears likely that <keygen> elements will be dropped from future versions of HTML so it may be desirable not to use <keygen> elements.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
`type`	Returns type of form element in which the keygen field appears	Here

The default style applicable to this element is shown here.

<label>

[HTMLElementLabel]

The HTML `<label>` element indicates a label for an <input> element. It does not appear special as far as the user is concerned, but does improve the usability of the <input> element, as it means that if the user clicks on the text within `<label>` element then it toggles the control.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`for`	Specifies which form element(s) a label calculation is bound to	Here
`form`	Name of the form that element belongs to (this should be the id attribute of the element to which the label element relates, to bind the two together)	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `for` property of the underlying element corresponding to the `htmlFor` property of the DOM object). It used to support the `control` property, but this was removed from the HTML specification in 2016.

The default style applicable to this element is shown here.

<legend>

[HTMLElementLegend]

The HTML `<legend>` element indicates a caption for a <fieldset> element.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to accept the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. It also supports the following additional properties:

Property	Description	More
form	Returns form that contains element	Here

The default style applicable to this element is shown here.

[HTMLElementLi]

The HTML <li> element indicates a list item. It is used in <ol> elements (ordered lists), <ul> elements (unordered lists) and <menu> elements (menu lists).

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
value	Value of element	Here

It used to support the type attribute (which specified what kind of bullet point should be used with the list element), but this is no longer supported in HTML 5 (similar effects can be achieved using CSS).

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<link>

[HTMLElementLink]

The HTML <link> element defines the relationship between a document and an external resource. It is most commonly used to link to CSS style sheets. It is an empty element (i.e. it only contains attributes) and should only appear in the <head> element of an HTML document (but can appear any number of times there).

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
crossorigin	Specifies how element handles cross-origin requests	Here
href	URL of page the link goes to	Here
hreflang	Language of linked document	Here
media	Specifies media / device linked document is optimised for	Here
rel	Relationship between current document and linked document	Here
sizes	Specifies size of linked resource	Here
type	Type of element	Here

It used to support the `charset`, `rev` and `target` attributes, but these are no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `crossorigin` property of the underlying element corresponding to the `crossOrigin` property of the DOM object). The default style applicable to this element is shown here.

<main>

[HTMLElementMain]

The HTML `<main>` element indicates the main content of a document. It is new in HTML 5. Content within the element should ideally be unique to the document and hence should not include material such as sidebars, copyright information, logos and search forms.

There shouldn't be more than one `<main>` element in a document and it shouldn't be a descendent of an <article>, <aside>, <footer>, <header> or <nav> element.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<map>

[HTMLElementMap]

The HTML `<map>` element indicates a client-side image-map (i.e. an image with clickable areas). It needs a name attribute, which creates a relationship between the image and the map. It typically contains one or more <area> elements that indicate which parts of the image map are clickable. In HTML 5, if the id attribute of the `<map>` element is specified then it needs to have the same value as the name attribute.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`name`	Name of associated element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
`areas`	Returns a collection of all <area> elements linked to the `<map>` element	Here
`images`	Returns a collection of all <img> and <object> elements	Here

	linked to the `<map>` element	

The default style applicable to this element is shown here.

<mark>

[HTMLElementMark]

The HTML `<mark>` element indicates marked/highlighted text. It is new in HTML 5.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<menu>

[HTMLElementMenu]

The HTML `<menu>` element indicates a menu or list of commands. In theory, it can be used for toolbars, context menus, listing form controls and commands. However, at the time of writing it was not supported by many browsers.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`label`	Title / label of command or group of commands	Here
`type`	Type of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<menuitem>

[HTMLElementMenuitem]

The HTML `<menuitem>` element indicates a menu item/command that a user can invoke from a popup menu. It is new in HTML 5. In theory, it can be used for toolbars, context menus, listing form controls and commands. However, at the time of writing it was not supported by many browsers.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`checked`	Specifies that the element should be pre-selected	Here
`default`	Default track / command to be enabled unless user preferences specify otherwise	Here
`disabled`	Specified element(s) to be disabled	Here

icon	Icon for a command / menu item	Here
label	Title / label command	Here
radiogroup	Name of group of commands when menu item toggled	Here
type	Type of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
command	Sets / returns the command property of the DOM object	Here

The default style applicable to this element is shown here.

<meta>

[HTMLElementMeta]

The HTML <meta> element indicates metadata about an HTML document. Metadata is not displayed on the page but is machine readable. The <meta> element always goes within the <head> element. Metadata within a <meta> element are always passed in pairs, one part describing what type of metadata is involved, the other indicating the value to ascribe to the metadata.

For example, the following are uses of the <meta> element:

Use to which a specific <meta> element can be put	Example
Page author	<meta name="author" content="Nematrian">
Page description	<meta name="description" content="HTML meta element">
Page keywords (used by search engines)	<meta name="keywords" content="HTML, Reference, Metadata">
Page to refresh every say 60 seconds	<meta http-equiv="refresh" content="60">
Page viewport that aims to make webpage look good on all devices	<meta name="viewport" content="width=device-width, initial-scale=1.0">

In HTML 5 a viewport metadata component was introduced (see above), allowing webpage designers greater control over the user's viewing experience, i.e. how the browser handles the browser window within which the page is viewed.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
charset	Specifies character encoding	Here
content	Value associated with the http-equiv or name attribute	Here
http-equiv	HTTP header for information/value of attribute	Here
name	Name of piece of metadata	Here

The element used to support the `scheme` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `http-equiv` property of the underlying element corresponding to the `httpEquiv` property of the DOM object). The default style applicable to this element is shown here.

<meter>

[HTMLElementMeter]

The HTML `<meter>` element indicates a scalar measurement within a specific range (otherwise called a gauge). It can also take fractional values. It is new in HTML 5. It is not designed to be used to show task progress, for which the preferred element is the <progress> element.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`form`	Name of the form that element belongs to	Here
`high`	Value above which is considered a high value	Here
`low`	Value below which is considered a low value	Here
`max`	Maximum value	Here
`min`	Minimum value	Here
`optimum`	Value deemed optimal for gauge	Here
`value`	Value of element	Here

The `high`, `low`, `max` and `min` attributes should satisfy: $min < low < high < max$. Not all major browsers currently support the `high` and `low` attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. It also supports the following additional properties and methods:

Property	Description	More
`labels`	Returns a collection of <label> elements corresponding to the gauge labels	Here

The default style applicable to this element is shown here.

<nav>

[HTMLElementNav]

The HTML `<nav>` element indicates navigation links. It is new in HTML 5. It is usual not to put all navigation links inside a `<nav>` element. Instead this element is intended only for major blocks of such links (so that browsers such as for disabled users can use this element to determine when to omit initial rendering of content).

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<noframes>

[HTMLElementNoframes]

The HTML <noframes> element was used to indicate alternate content for users whose browsers do not support frames. It is not supported in HTML 5.

<noscript>

[HTMLElementNoscript]

The HTML <noscript> element indicates the alternate content to be used for users whose browsers do not support client-side scripts (either because the browser doesn't support them, which is rare these days, or because users have disabled their use). It can be used inside both <head> and <body> elements. With the former, it can only contain <link>, <style> and <meta> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<object>

[HTMLElementObject]

The HTML <object> element indicates an embedded object, such as a Java applet, ActiveX or Flash plugin. It can also be used to embed another webpage into the HTML document. <param> elements can be used to pass parameters to plugins embedded within <object> elements.

<object> elements must appear inside the <body> element of the webpage. Text between the opening <object> and the closing </object> tags is interpreted as alternative text that is displayed for browsers that do not support the <object> element. At least one of element's data or type attributes needs to be defined.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
data	URL of resource to be used by object	Here
form	Name of the form that element belongs to	Here
height	Height of element	Here
name	Name of element	Here
type	Type of element	Here
usemap	Specifies an image as a client-side image-map	Here
width	Width of element	Here

It used to support the `align`, `archive`, `border`, `classid`, `codebase`, `codetype`, `declare`, `hspace`, `standby` and `vspace` attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `usemap` property of the underlying element corresponding to the `useMap` property of the DOM object).

The default style applicable to this element is shown here.

[HTMLElementOl]

The HTML `<ol>` element indicates an ordered list. The list can be numerical or alphabetical. Individual items within the list are identified using <li> elements. Lists can be styled using CSS.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`reversed`	List order should be descending (3, 2, 1) not ascending (1, 2, 3)	Here
`start`	Start value of an ordered list	Here
`type`	Type of element	Here

It used to support the `compact` attribute, but this is no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<optgroup>

[HTMLElementOptgroup]

The HTML `<optgroup>` element indicates a group of related options in a drop-down list.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`disabled`	Specified element(s) (here the option-group) to be disabled	Here
`label`	Title of option group	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above). The default style applicable to this element is shown here.

<option>

[HTMLElementOption]

The HTML `<option>` element indicates an option in a drop-down list.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`disabled`	Specified element(s) (here the option-group) to be disabled	Here
`label`	Title of option group	Here
`selected`	Indicates that an <option> element should be pre-selected when the page loads	Here
`value`	Value of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
`defaultSelected`	Returns default value of the selected attribute	Here
`form`	Returns reference to form that contains the option	Here
`index`	Sets / returns index position of option in drop-down list	Here
`text`	Sets / returns text of the option	Here

The default style applicable to this element is shown here.

<output>

[HTMLElementOutput]

The HTML `<output>` element indicates the result of a calculation (e.g. one performed by a script). It is new in HTML 5.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`for`	Specifies which form element(s) a label calculation is bound to	Here
`form`	Name of the form that element belongs to	Here
`name`	Name of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `for` property

of the underlying element corresponding to the `htmlFor` property of the DOM object). It also supports the following additional properties:

Property	Description	More
`defaultValue`	Sets / returns default value	Here
`labels`	Returns a collection of <label> elements associated with the `<output>` object	Here
`type`	Returns type of HTML element represented by the `<output>` object	Here
`value`	Sets / returns value of element	Here

The default style applicable to this element is shown here.

<p>

[HTMLElementP]

The HTML `<p>` element indicates a paragraph.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align` attribute, but this is no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<param>

[HTMLElementParam]

The HTML `<param>` element indicates a parameter for an object.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`name`	Name of associated element	Here
`value`	Value of element	Here

It used to support the `type` and `valuetype` attributes, but these are no longer supported in HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above). The default style applicable to this element is shown here.

<picture>

[HTMLElementPicture]

The HTML <picture> element indicates a container for multiple image resources. A <picture> element contains zero or more <source> elements followed by one <img> element. The source element(s) will be differentiated by different srcset attributes (required, defines the URL of the image to be shown by the <picture> element) and by the media attribute (optional, a CSS media query that identifies which media relates to that URL). The browser uses the first matching <source> element, and if none match then it defaults to the <img> element.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<pre>

[HTMLElementPre]

The HTML <pre> element indicates a piece of preformatted text. Typically the text is displayed in a fixed-width font (usually Courier), preserving both spaces and line breaks.

The attributes it can take are HTML global attributes and HTML event attributes. It used to support the width attribute, but this is no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<progress>

[HTMLElementProgress]

The HTML <progress> element is most commonly used to show the progress of some task. It is new in HTML 5. It is not very suitable for representing a gauge (like a fuel tank), for which better usually is to use a <meter> element.

For example, markup as follows:

```
Progress so far: <progress value="40" max="100">
```

creates output that involves a progress bar showing that 40% of the task has been completed:

If you want the bar to be narrower than it is by default then you need to use the width attribute within the style of the element, e.g. markup as follows:

```
Progress so far: <progress value="33" max="100" style="width:40px">
```

Usually a progress bar will be updated as time progresses, often using JavaScript.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
max	Maximum value	Here
value	Value of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
labels	Returns a list of the progress bar labels (if any)	Here
position	Returns current position of progress bar	Here

The default style applicable to this element is shown here.

<q>

[HTMLElementQ]

The HTML <q> element indicates a short quotation.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
cite	URL which explains the quote	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<rp>

[HTMLElementRp]

The HTML <rp> element indicates what to show in browsers that do not support ruby annotations (for East Asian typography). It is new in HTML 5.

It is used in conjunction with <rt> and <ruby> elements (the <ruby> element includes one or more characters that need an explanation / pronunciation, the <rt> element gives that information and the optional <rp> element indicates what to show for browsers that do not support such characters.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<rt>

[HTMLElementRt]

The HTML <rt> element indicates an explanation / pronunciation of characters (typically for East Asian typography). It is new in HTML 5.

It is used in conjunction with <rp> and <ruby> elements (the <ruby> element includes one or more characters that need an explanation / pronunciation, the <rt> element gives that information and the optional <rp> element indicates what to show for browsers that do not support such characters.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<ruby>

[HTMLElementRuby]

The HTML <ruby> element indicates ruby annotation (for East Asian typography). It is new in HTML 5.

It is used in conjunction with <rp> and <rt> elements (the <ruby> element includes one or more characters that need an explanation / pronunciation, the <rt> element gives that information and the optional <rp> element indicates what to show for browsers that do not support such characters.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<s>

[HTMLElementStrikeThrough]

The HTML <s> element indicates text that is no longer correct. Conventionally, the <s> element should not be used for replaced or deleted text (instead use a <del> element).

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<samp>

[HTMLElementSamp]

The HTML <samp> element is a phrase element indicating sample output from a computer program. It is not depreciated, but typically a richer effect can be achieved using CSS.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<script>

[HTMLElementScript]

The HTML <script> element indicates client-side script / programming code. Usually this is written in JavaScript. The <script> element either contains this code or points to an external file via its src attribute (if the src attribute is present then the <script> element must be empty). The contents of an noscript element indicates what happens for users who have disabled scripts in their browser or whose browser does not support client-side scripting.

The way in which a script executes is driven by the async and defer attributes. If neither are present then the script is fetched and executed immediately the browser reaches the element (before the browser continues parsing the page).

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`async`	Indicates if script to be executed asyncronously	Here. Only for external scripts
`charset`	Specifies character encoding	Here
`defer`	Script to be executed only when page has finished parsing	Here. Only for external scripts
`src`	URL of resource	Here
`type`	Type of element	Here

It used to support the `xml:space` attribute, but this is no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties:

Property	Description	More
`crossOrigin`	Sets / returns the CORS settings for the script	Here
`text`	Sets / returns contents of all child text nodes of the script	Here

The default style applicable to this element is shown here.

<section>

[HTMLElementSection]

The HTML `<section>` element indicates a section in a document (e.g. a discrete chapter / heading / footer etc>. It is new in HTML 5.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<select>

[HTMLElementSelect]

The HTML `<select>` element indicates a drop-down list. The option elements within the `<select>` element identify the available options within the drop-down list

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`autofocus`	Specifies whether element should automatically get focus when page loads	Here
`disabled`	Specified element(s) to be disabled	Here
`form`	Name of the form that element belongs to	Here
`multiple`	Indicates that a user can enter more than one value	Here
`name`	Name of element	Here
`required`	Whether the element must be filled out before submitting form	Here
`size`	Specifies number of visible options	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties and methods:

Additional properties:

Property	Description	More
`length`	Returns number of option elements within the drop-down list	Here
`options`	Returns a collection of all options in drop-down list	Here
`selectedIndex`	Sets / returns index of selected option	Here
`type`	Returns type of form the drop-down list is within	Here
`value`	Sets / returns the value of the selected option in the drop-down list	Here

Additional methods:

Method	Description	More
`add()`	Adds an option to drop-down list	Here
`remove()`	Removes an option from drop-down list	Here

The default style applicable to this element is shown here.

<small>

[HTMLElementSmall]

The HTML `<small>` element indicates smaller text.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<source>

[HTMLElementSource]

The HTML `<source>` element allows multiple media resources for media elements. It links together associated <video> and <audio>. It is new in HTML 5. The srcset attribute is required if the `<source>` element is used in a picture element, whilst the src attribute is required when the `<source>` element is used in an <audio> or <video> element.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`media`	Specifies media / device linked document is optimised for	Here
`sizes`	Specifies image size(s)	Here
`src`	Required when URL of resource	Here
`srcset`	URL of image to use in different situations	Here
`type`	Type of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above). The default style applicable to this element is shown here.

[HTMLElementSpan]

The HTML `<span>` element indicates a section in a document. It is usually defined with its own style.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<strike>

[HTMLElementStrike]

The HTML `<strike>` element was used to indicate strikethrough text. It is not supported in HTML 5. Instead, use the <del> or <s> element.

[HTMLElementStrong]

The HTML `<strong>` element is a phrase element indicating more important text. It is commonly used as a way of highlighting text or making it bold.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<style>

[HTMLElementStyle]

The HTML `<style>` element indicates style information for a document. HTML documents can contain multiple `<style>` elements. See CSS Tutorial for further details.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`media`	Specifies media / device linked document is optimised for	Here
`scoped`	Indicates styles only apply to the element's parent element and that element's child elements	Here
`type`	Type of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object (typically the style property of an element) supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties, see also CSS Properties:

DOM Property Name	Corresponding CSS Property Name	More
`alignContent`	`align-content`	Here
`alignItems`	`align-items`	Here
`alignSelf`	`align-self`	Here
`animation`	`animation`	Here

animationDelay	animation-delay	Here
animationDirection	animation-direction	Here
animationDuration	animation-duration	Here
animationFillMode	animation-fill-mode	Here
animationIterationCount	animation-iteration-count	Here
animationName	animationName	Here
animationPlayState	animationPlayState	Here
animationTimingFunction	animationTimingFunction	Here
backfaceVisibility	backface-visibility	Here
background	background	Here
backgroundAttachment	background-attachment	Here
backgroundClip	background-clip	Here
backgroundColor	background-color	Here
backgroundImage	background-image	Here
backgroundOrigin	background-origin	Here
backgroundPosition	background-position	Here
backgroundRepeat	background-repeat	Here
backgroundSize	background-size	Here
border	border	Here
borderBottom	border-bottom	Here
borderBottomColor	border-bottom-color	Here
borderBottomLeftRadius	border-bottom-left-radius	Here
borderBottomRightRadius	border-bottom-right-radius	Here
borderBottomStyle	border-bottom-style	Here
borderBottomWidth	border-bottom-width	Here
borderCollapse	border-collapse	Here
borderColor	border-color	Here
borderImage	border-image	Here
borderImageOutset	border-image-outset	Here
borderImageRepeat	border-image-repeat	Here
borderImageSlice	border-image-slice	Here
borderImageSource	border-image-source	Here
borderImageWidth	border-image-width	Here
borderLeft	border-left	Here
borderLeftColor	border-left-color	Here
borderLeftStyle	border-left-style	Here
borderLeftWidth	border-left-width	Here
borderRadius	border-radius	Here
borderRight	border-right	Here
borderRightColor	border-right-color	Here
borderRightStyle	border-right-style	Here
borderRightWidth	border-right-width	Here
borderSpacing	border-spacing	Here
borderStyle	border-style	Here
borderTop	border-top	Here
borderTopColor	border-top-color	Here
borderTopLeftRadius	border-top-left-radius	Here
borderTopRightRadius	border-top-right-radius	Here
borderTopStyle	border-top-style	Here
borderTopWidth	border-top-width	Here

borderWidth	border-width	Here
bottom	bottom	Here
boxShadow	box-shadow	Here
boxSizing	box-sizing	Here
captionSide	caption-side	Here
clear	clear	Here
clip	clip	Here
color	color	Here
columnCount	column-count	Here
columnFill	column-fill	Here
columnGap	column-gap	Here
columnRule	column-rule	Here
columnRuleColor	column-rule-color	Here
columnRuleStyle	column-rule-style	Here
columnRuleWidth	column-rule-width	Here
columnSpan	column-span	Here
columnWidth	column-width	Here
columns	columns	Here
content	content	Here
counterIncrement	counter-increment	Here
counterReset	counter-reset	Here
cursor	cursor	Here
direction	direction	Here
display	display	Here
emptyCells	empty-cells	Here
filter	filter	Here
flex	flex	Here
flexBasis	flex-basis	Here
flexDirection	flex-direction	Here
flexFlow	flex-flow	Here
flexGrow	flex-grow	Here
flexShrink	flex-shrink	Here
flexWrap	flex-wrap	Here
cssFloat	float	Here
font	font	Here
fontFamily	font-family	Here
fontSize	font-size	Here
fontSizeAdjust	font-size-adjust	Here
fontStretch	font-stretch	Here
fontStyle	font-style	Here
fontVariant	font-variant	Here
fontWeight	font-weight	Here
hangingPunctuation	hanging-punctuation	Here
height	height	Here
justifyContent	justify-content	Here
left	left	Here
letterSpacing	letter-spacing	Here
lineHeight	line-height	Here
listStyle	list-style	Here
listStyleImage	list-style-image	Here

listStylePosition	list-style-position	Here
listStyleType	list-style-type	Here
margin	margin	Here
marginBottom	margin-bottom	Here
marginLeft	margin-left	Here
marginRight	margin-right	Here
marginTop	margin-top	Here
maxHeight	max-height	Here
maxWidth	max-width	Here
minHeight	min-height	Here
minWidth	min-width	Here
navDown	nav-down	Here
navIndex	nav-index	Here
navLeft	nav-left	Here
navRight	nav-right	Here
navUp	nav-up	Here
opacity	opacity	Here
order	order	Here
orphans	orphans	Here
outline	outline	Here
outlineColor	outline-color	Here
outlineOffset	outline-offset	Here
outlineStyle	outline-style	Here
outlineWidth	outline-width	Here
overflow	overflow	Here
overflowX	overflow-x	Here
overflowY	overflow-y	Here
Padding	padding	Here
paddingBottom	padding-bottom	Here
paddingLeft	padding-left	Here
paddingRight	padding-right	Here
paddingTop	padding-top	Here
pageBreakAfter	page-break-after	Here
pageBreakBefore	page-break-before	Here
pageBreakInside	page-break-inside	Here
perspective	perspective	Here
perspectiveOrigin	perspective-origin	Here
position	position	Here
quotes	quotes	Here
resize	resize	Here
right	right	Here
tabSize	tab-size	Here
tableLayout	table-layout	Here
textAlign	text-align	Here
textAlignLast	text-align-last	Here
textDecoration	text-decoration	Here
textDecorationColor	text-decoration-color	Here
textDecorationLine	text-decoration-line	Here
textDecorationStyle	text-decoration-style	Here
textIndent	text-indent	Here

textJustify	text-justify	Here
textOverflow	text-overflow	Here
textShadow	text-shadow	Here
textTransform	text-transform	Here
top	top	Here
transform	transform	Here
transformOrigin	transform-origin	Here
transformStyle	transform-style	Here
transition	transition	Here
transitionDelay	transition-delay	Here
transitionDuration	transition-duration	Here
transitionProperty	transition-property	Here
transitionTimingFunction	transition-timing-function	Here
unicodeBidi	unicode-bidi	Here
userSelect	user-select	Here
verticalAlign	vertical-align	Here
visibility	visibility	Here
whiteSpace	white-space	Here
widows	widows	Here
width	width	Here
wordBreak	word-break	Here
wordSpacing	word-spacing	Here
wordWrap	word-wrap	Here
zIndex	z-index	Here

The default style applicable to this element is shown here.

<sub>

[HTMLElementSub]

The HTML <sub> element indicates subscripted text.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<summary>

[HTMLElementSummary]

The HTML <summary> element indicates a heading for a <details> element. It is new in HTML 5.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<sup>

[HTMLElementSup]

The HTML <sup> element indicates superscripted text.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<table>

[HTMLElementTable]

The HTML <table> element indicates a table. It typically includes one or more <tr> elements and, within them, <td> and/or <th> elements. More complicated table layouts can also include <caption>, <col>, <colgroup>, <tbody>, <tfoot> and <thead> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align`, `bgcolor`, `border`, `cellpadding`, `cellspacing`, `frame`, `rules`, `summary` and `width` attributes, but these are no longer supported by HTML 5. Original draft versions of HTML 5 also included a `sortable` attribute, but this appears to have been dropped in later specifications and not to have been implemented so far by major browsers.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. It also supports the following additional properties and methods:

Additional properties:

Property	Description	More
`caption`	Returns the <caption> element of the table	Here
`rows`	Returns a collection of the <tr> elements of the table	Here
`tBodies`	Returns a collection of <tbody> elements of the table	Here
`tFoot`	Returns the <tfoot> element of the table	Here
`tHead`	Returns the <thead> element of the table	Here

Additional methods:

Method	Description	More
`createCaption()`	Creates empty <caption> element and adds to table	Here
`createTFoot()`	Creates empty <tfoot> element and adds to table	Here
`createTHead()`	Creates empty <thead> element and adds to table	Here
`deleteCaption()`	Removes first <caption> element from table	Here
`deleteRow()`	Removes a <tr> element from table	Here
`deleteTFoot()`	Removes first <tfoot> element from table	Here

deleteTHead()	Removes <thead> element from table	Here
insertRow()	Creates empty <tr> element and adds to table	Here

The default style applicable to this element is shown here.

<tbody>

[HTMLElementTbody]

The HTML <tbody> element indicates the body of a table. It appears inside a <table> element and is used in conjunction with <tfoot> and <thead> elements to differentiate between different parts of the table. This can allow browsers to scroll the table body independently of the header and footer, or to allow printing of the header and footer at the top and bottom of each page. A <tbody> element needs to come after any <caption>, <colgroup> and <thead> elements. It needs to contain one or more <tr> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the align, char, charoff and valign attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<td>

[HTMLElementTd]

The HTML <td> element indicates a table cell (within a table row). They appear inside <tr> elements. HTML tables contain two types of cells, i.e. header cells (<th> elements) and standard cells (<td> elements), and the two are by default formatted differently.

The attributes it can take (in addition to HTML global attributes and HTML event attributes) are:

Attribute	Description	More
colspan	Number of columns a table cell should span	Here
headers	One or more header cells a cell is related to	Here
rowspan	Number of rows a table cell should span	Here

It used to support the abbr, align, axis, bgcolor, char, charoff, height, nowrap, scope, valign and width attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<textarea>

[HTMLElementTextarea]

The HTML `<textarea>` element indicates a multiline input control. It can hold an unlimited number of characters, and the text used is typically rendered in a fixed-width font. The size of the text area can be specified using the element's `cols` and `rows` attributes or using corresponding CSS attributes.

The attributes it can take (other than HTML global attributes and HTML event attributes) include:

Attribute	Description	More
`autofocus`	Specifies whether element should automatically get focus when page loads	Here
`cols`	Width of text area (in characters)	Here
`dirname`	Specifies text direction will be submitted	Here
`disabled`	Specified element(s) to be disabled	Here
`form`	Name of the form that element belongs to	Here
`maxlength`	Maximum number of characters allowed in an element	Here
`name`	Name of element	Here
`placeholder`	Short hint describing expected value of element	Here
`readonly`	Whether element is read-only	Here
`required`	Whether the element must be filled out before submitting form	Here
`rows`	Visible number of lines in a <textarea> element	Here
`wrap`	How text in a <textarea> element is to be wrapped when submitted in a form	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above (with the `maxlength` and `readonly` properties of the underlying element corresponding to the `maxLength` and `readOnly` properties of the DOM object). It also supports the following additional properties and methods:

Additional properties:

Property	Description	More
`defaultValue`	Sets / returns default value of element	Here
`type`	Returns type of form that contains element	Here
`value`	Sets / returns contents of element	Here

Additional methods:

Method	Description	More
`select()`	Selects entire contents of text area	Here

The default style applicable to this element is shown here.

<tfoot>

[HTMLElementTfoot]

The HTML <tfoot> element indicates the footer content in a table. It appears inside a <table> element and is used in conjunction with <tbody> and <thead> elements to differentiate between different parts of the table. This can allow browsers to scroll the table body independently of the header and footer, or to allow printing of the header and footer at the top and bottom of each page. A <tfoot> element needs to come after any <caption>, <colgroup> and <thead> elements and before any <tbody> and <tr> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the align, char, charoff and valign attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<th>

[HTMLElementTh]

The HTML <th> element indicates a table header cell (within a table row). They appear inside <tr> elements. HTML tables contain two types of cells, i.e. header cells (<th> element) and standard cells (<td> elements), and the two are by default formatted differently.

The attributes it can take (in addition to HTML global attributes and HTML event attributes) are:

Attribute	Description	More
colspan	Number of columns a table cell should span	Here
headers	One or more header cells a cell is related to	Here
rowspan	Number of rows a table cell should span	Here

It used to support the abbr, align, axis, bgcolor, char, charoff, height, nowrap, scope, valign and width attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<thead>

[HTMLElementThead]

The HTML <thead> element indicates the header content of a table. It appears inside a <table> element and is used in conjunction with <tbody> and <tfoot> elements to differentiate between different parts of the table. This can allow browsers to scroll the table body independently of the header and footer, or to allow printing of the header and footer at the top and bottom of each page. A <thead> element needs to come after any <caption> and <colgroup> elements and before any <tbody>, <tfoot> and <tr> elements.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align`, `char`, `charoff` and `valign` attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<time>

[HTMLElementTime]

The HTML `<time>` element indicates a (human-readable) date / time. It can be used to encode dates and times in a machine-readable fashion.

The attributes it can take (in addition to HTML global attributes and HTML event attributes) are:

Attribute	Description	More
`datetime`	Date and time of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. The default style applicable to this element is shown here.

<title>

[HTMLElementTitle]

The HTML `<title>` element indicates the title for the document. It appears in the <head> part of the document. It typically identifies the page title that appears in a browser toolbar, the page title that is by default added to a user's list of favourite pages within a browser and usually is the title shown for the page in search engine results.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. It also supports the following additional properties:

Property	Description	More
`text`	Sets / returns text of document title	Here

The default style applicable to this element is shown here.

<tr>

[HTMLElementTr]

The HTML `<tr>` element indicates a table row (within a table). It appears inside a <table> element and contains <td> and <th> elements representing individual cells within the table row.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the `align`, `bgcolor`, `char`, `charoff` and `valign` attributes, but these are no longer supported in HTML5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. It also supports the following additional properties and methods:

Additional properties:

Property	Description	More
`cells`	Returns collection of all <td> and <th> elements in row	Here
`rowIndex`	Returns position of row in rows collection of a <table> element	Here
`sectionRowIndex`	Returns position of row in rows collection of a <tbody>, <tfoot> or a <thead>	Here

Additional methods:

Method	Description	More
`deleteCell()`	Deletes a cell from table row	Here
`insertCell()`	Inserts a cell into table row	Here

The default style applicable to this element is shown here.

<track>

[HTMLElementTrack]

The HTML `<track>` element indicates a text track for a media element, i.e. a <video> or <audio>. It is new in HTML 5. It is used to specify subtitles, captions or other files containing text that should be visible when the media is playing.

The attributes it can take (in addition to HTML global attributes and HTML event attributes) are:

Attribute	Description	More
`default`	Default track / command to be enabled unless user preferences specify otherwise	Here
`kind`	Kind of text track	Here
`label`	Title / label of track or command or group of commands	Here
`src`	URL of track file	Here
`srclang`	Language of track text data	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports the following additional properties and methods:

Property	Description	More
readyState	Returns current state of track resource	Here
track	Returns TextTrack object representing the text track data of the track element	Here

The default style applicable to this element is shown here.

<tt>

[HTMLElementTt]

The HTML <tt> element indicates teletype text. It is not supported in HTML 5. Instead, use CSS.

<u>

[HTMLElementU]

The HTML <u> element indicates text that should be stylistically different from normal text. It is commonly used for underlining, even though the HTML 5 specification reminds developers that there are almost always other more appropriate ways of achieving a similar effect.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

[HTMLElementUl]

The HTML <ul> element indicates an unordered list. Inside the <ul> element should be one or more <li> elements identifying each entry in the unordered list.

The attributes it can take are HTML global attributes and HTML event attributes.

It used to support the compact and type attributes, but these are no longer supported by HTML 5.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<var>

[HTMLElementVar]

The HTML <var> element is a phrase element indicating a variable in computer code. It is not depreciated, but typically a richer effect can be achieved using CSS.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

<video>

[HTMLElementVideo]

The HTML `<video>` element indicates a video or movie. It is new in HTML 5. Currently there are 3 supported video formats across most browsers: MP4 (i.e. MPEG 4 files with H264 video codec and AAC audio codec, MIME-type is video/mp4), WebM (i.e. WebM files with V8 video codec and Vorbis audio codec, MIME-type is video/webm) and Ogg (Ogg files with Theora video codec and Vorbis audio codec, MIME-type is video/ogg).

If the browser does not support `<video>` elements then any text between the `<video>` and `</video>` tags will be displayed.

The attributes it can take (in addition to HTML global attributes and HTML event attributes) are:

Attribute	Description	More
`autoplay`	Specifies whether media should start playing as soon as ready	Here
`controls`	Whether controls (such as play and pause buttons) should be displayed	Here
`height`	Height of element	Here
`loop`	Media to start over again when it finishes	Here
`muted`	Audio output should be muted	Here
`poster`	Image to be shown while video is downloading (or until user hits play)	Here
`preload`	If / how author thinks media should be loaded when page loads	Here
`src`	URL of media	Here
`width`	Width of element	Here

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods, and additional properties with the same name and meaning as the attributes of the underlying HTML element referred to above. It also supports DOM generic media properties and methods and the following additional properties and methods.

Additional properties:

Property	Description	More
`videoTracks`	Returns VideoTrackList object indicating available video tracks	Here

The default style applicable to this element is shown here.

<wbr>

[HTMLElementWbr]

The HTML <wbr> element indicates a possible line-break (i.e. 'word break opportunity'). It is new in HTML 5. When a word is long and the browser might break text lines in wrong places then <wbr> elements offer scope to add word break opportunities.

The attributes it can take are HTML global attributes and HTML event attributes.

To create or access such an element in JavaScript see here. The corresponding HTML DOM object supports standard DOM properties and methods. The default style applicable to this element is shown here.

Appendix B: HTML Attributes

[HTMLAttributes]

Different HTML elements can have attributes that specify how they should be formatted or interpreted or allow further characterisation of the element. Attributes come in two basic types:

(a) Standard attributes, which describe or characterise the element further, and
(b) Event attributes, almost all of which begin with `on`…. These indicate what scripts should be run if an *event* occurs (e.g. the mouse button is clicked, an element is dragged, dropped or copied, etc.)

HTML attributes can also be set programmatically using JavaScript by modifying the properties of the corresponding HTML DOM elements. Listed here are the JavaScript DOM properties that correspond to most of the HTML attributes recognised in HTML 5.

Standard attributes

[HTMLStandardAttributes]

Different HTML elements can have attributes that specify how they should be formatted or interpreted or allow further characterisation of the element, which can be either standard attributes or event attributes. Standard attributes describe or characterise the element further and include:

Attribute	Description	Applicable to	More
`accept`	Specifies types of file accepted by server	<input>	Here
`accept-charset`	Specifies character encodings used for form submission	<form>	Here
`accesskey`	Shortcut key to activate/focus element	All	Here
`action`	Where to send form-data when form submitted	<form>	Here
`align`	Alignment versus surrounding elements		Here. Not supported in HTML 5 (instead use CSS)
`alt`	Specifies alternative text to show when original content fails to display	Various	Here
`async`	Indicates if script to be executed asyncronously	<script>	Here. Only for external scripts
`autocomplete`	Specifies whether element has autocomplete enabled	<form>, <input>	Here
`autofocus`	Specifies whether element should automatically get focus when page loads	Various	Here
`autoplay`	Specifies whether audio/video should start playing as soon as ready	<audio>, <video>	Here

bgcolor	Specifies background colour		Here. Not supported in HTML 5 (instead use CSS)
border	Specifies width of border		Here. Not supported in HTML 5 (instead use CSS)
challenge	Indicates value of element should be challenged when submitted	<keygen>	Here
charset	Specifies character encoding	<meta>, <script>	Here
checked	Specifies that the element should be pre-selected	<input>, <menuitem>	Here
cite	URL which explains the quote / deleted / inserted text	<blockquote>, <del>, <ins>, <q>	Here
class	One or more class names for the element	All	Here. Refers to a class in a CSS style
color	Text colour of element	All text elements	Here. Not supported in HTML 5 (instead use CSS)
cols	Width of text area (in characters)	<textarea>	
colspan	Number of columns a table cell should span	<td>, <th>	Here
content	Value associated with the http-equiv or name attribute	<meta>	Here
contenteditable	Indicates whether content is editable	All	Here
contextmenu	Specifies context menu (i.e. what appears when right-click)	All	Here
controls	Whether <audio> and <video> controls (such as play and pause buttons) should be displayed	<audio>, <video>	Here
coords	Specifies the coordinates of an <area>	<area>	Here
crossorigin	Specifies how element handles cross-origin requests	<img>, <link>	Here
data	URL of resource to be used by object	<object>	Here
data-*	Custom data private to page of application	All	Here
datetime	Date and time of element	<del>, <ins>, <time>	Here
default	Default track / command to be enabled unless user preferences specify otherwise	<menuitem>, <track>	Here
defer	Script to be executed only	<script>	Here. Only for

	when page has finished parsing		external scripts
dir	Text direction for element content	All	Here
dirname	Specifies text direction will be submitted	<input>, <textarea>	Here
disabled	Specified element(s) to be disabled	Various	Here
download	Target will be downloaded when user clicks on hyperlink	<a>, <area>	Here
draggable	Whether element is draggable	All	Here
dropzone	Whether dragged data is copied, moved or linked when dropped	All	Here
enctype	How form-data to be encoded when submitted to server	<form>	Here. Only for method = post
for	Specifies which form element(s) a label calculation is bound to	<label>, <output>	Here
form	Name of the form that element belongs to	Various	Here
formaction	Where to send form-data to when form submitted	<button>, <input>	Here. Only for type = submit
formenctype	How form-data should be encoded before sending it to a server	<button>, <input>	Here. Only for type = submit
formmethod	How to send form-data (i.e. which HTTP method to use)	<button>, <input>	Here. Only for type = submit
formnovalidate	Specifies that form-data should not be validated on submission	<button>, <input>	Here. Only for type = submit
formtarget	Specifies where to display the response that is received after submitting form	<button>, <input>	Here. Only for type = submit
headers	One or more header cells a cell is related to	<td>, <th>	Here
height	Height of element	Various (not all)	Here
hidden	Whether element is not relevant	All	Here
high	Value above which is considered a high value	<meter>	Here
href	URL of page the link goes to	<a>, <area>, <base>, <link>	Here
hreflang	Language of linked document	<a>, <area>, <link>	Here
http-equiv	HTTP header for information/value of attribute	<meta>	Here

icon	Icon for a command / menu item	<menuitem>	Here
id	Unique id for an element (for e.g. JavaScript)	All	Here
ismap	Image is a server-side image-map	<img>	Here
keytype	Specifies security algorithm of key	<keygen>	Here
kind	Kind of text track	<track>	Here
label	Title / label of track or command or group of commands	<menu>, <menuitem>, <option>, <optgroup>, <track>	Here
lang	Language of an element's content	All	Here
list	Refers to <datalist> that contains pre-defined options	<input>	Here
loop	Audio / video to start over again when it finishes	<audio>, <video>	Here
low	Value below which is considered a low value	<meter>	Here
manifest	Specifies address of document's cache manifest (for offline browsing)	<html>	Here
max	Maximum value	<input>, <meter>, <progress>	Here
maxlength	Maximum number of characters allowed in an element	<input>, <textarea>	Here
media	Specifies media / device linked document is optimised for	<a>, <area>, <link>, <source>, <style>	Here
method	Specifies HTTP method used when sending form-data	<form>	Here
min	Minimum value	<input>, <meter>	Here
multiple	Indicates that a user can enter more than one value	<input>, <select>	Here
muted	Audio output should be muted	<audio>, <video>	Here
name	Name of element (or of a piece of metadata for a <meta> element)	Various	Here
novalidate	Form should not be validated when submitted	<form>	Here
open	Whether details should be visible (i.e. open) to user	<details>	Here
optimum	Value deemed optimal for gauge	<meter>	Here
pattern	Regular expression that value of element is checked	<input>	Here

	against		
placeholder	Short hint describing expected value of element	<input>, <textarea>	Here
poster	Image to be shown while video is downloading (or until user hits play)	<video>	Here
preload	If / how author thinks audio / video should be loaded when page loads	<audio>, <video>	Here
radiogroup	Name of group of commands when menu item toggled	<menuitem>	Here
readonly	Whether element is read-only	<input>, <textarea>	Here
rel	Relationship between current document and linked document	<a>, <area>, <link>	Here
required	Whether the element must be filled out before submitting form	<input>, <select>, <textarea>	Here
reversed	List order should be descending (3, 2, 1) not ascending (1, 2, 3)		Here
rows	Visible number of lines in a <textarea> element	<textarea>	Here
rowspan	Number of rows a table cell should span	<td>, <th>	Here
sandbox	Allows an extra set of restrictions for the content of an <iframe> element	<iframe>	Here
scope	Indicates whether a header cell is a header for a column, row or groups of these	<th>	Here
scoped	Indicates styles only apply to the element's parent element and that element's child elements	<style>	Here
selected	Indicates that an <option> element should be pre-selected when the page loads	<option>	Here
shape	Specifies shape of an <area> element	<area>	Here
size	Specifies width in characters for <input> or number of visible options for <select>	<input>, <select>	Here
sizes	Specifies size of linked resource	<img>, <link>, <source>	Here
span	Number of columns to span	<col>, <colgroup>	Here
spellcheck	Indicates whether element is to have its spelling and	All	Here

	grammar checked		
src	URL of resource	Various	Here
srcdoc	HTML content of page to show in an <iframe>	<iframe>	Here
srclang	Language of track text data	<track>	Here. New in HTML 5, for kind = subtitles
srcset	URL of image to use in different situations	<img>, <source>	Here
start	Start value of an ordered list		Here
step	Accepted number intervals for an <input> element	<input>	Here
style	Inline CSS style for an element	All	Here
tabindex	Tab order of an element	All	Here
target	Specifies where / how to open the linked document (or where to submit the form)	<a>, <area>, <base>, <form>	Here
title	Extra information about element	All	Here
translate	Whether content of an element should be translated	All	Here
type	Type of element	Various	Here
usemap	Specifies an image as a client-side image-map	<img>, <object>	Here
value	Value of element	Various	Here
width	Width of element	Various (not all)	Here
wrap	How text in a <textarea> element is to be wrapped when submitted in a form	<textarea>	Here
xmlns	XML namespace attribute applicable to the webpage (if it needs to conform to XHTML)	<html>	Here

Some standard attributes can apply to essentially all HTML elements. These are called global attributes.

Event attributes

[HTMLEventAttributes]

Different HTML elements can have attributes that specify how they should be formatted or interpreted or allow further characterisation of the element, which can be either standard attributes or event attributes. Event attributes indicate what scripts should be run if an event occurs (e.g. the mouse button is clicked, an element is dragged, dropped or copied, etc). HTML5 added many more possible event attributes that can be assigned to HTML elements. In each case the value of the attribute is a script to be run when an event occurs.

HTML event attributes include:

Event Attribute	Description of event	Applicable to	More
`animationend`	When CSS animation ends	Any element with a CSS animation	Here
`animationiteration`	When CSS animation is repeated	Any element with a CSS animation	Here
`animationstart`	When CSS animation starts	Any element with a CSS animation	Here
`onabort`	If document or media loading is aborted	<audio>, <embed>, <input> (if type = `image`), <img>, <object>, <video>	Here
`onafterprint`	After document printed	<body>	Here
`onbeforeprint`	Before document printed	<body>	Here
`onbeforeunload`	Just before document unloaded	<body>	Here
`onblur`	When element loses focus	All visible elements	Here
`oncanplay`	When file ready to start playing (i.e. when buffered enough to begin)	<audio>, <embed>, <object>, <video>	Here
`oncanplaythrough`	When file ready to be played all way to end without pausing for buffering	<audio>, <video>	Here
`onchange`	When element value changed	All visible elements	Here
`onclick`	When element clicked	All visible elements	Here
`oncontextmenu`	When context menu triggered	All visible elements	Here
`oncopy`	When content of element being copied	All visible elements	Here
`oncuechange`	When cue changes	<track>	Here
`oncut`	When content of element being cut	All visible elements	Here
`ondblclick`	When element is double-clicked	All visible elements	Here
`ondrag`	When element being dragged	All visible elements	Here
`ondragend`	At end of drag operation	All visible elements	Here
`ondragenter`	When element has been dragged to a valid drop target	All visible elements	Here
`ondragleave`	When element leaves a valid drop target	All visible elements	Here
`ondragover`	When element being dragged over a valid drop target	All visible elements	Here
`ondragstart`	At start of drag operation	All visible elements	Here

ondrop	When dragged element is being dropped	All visible elements	Here
ondurationchange	When length of media changes	<audio>, <video>	Here
onemptied	When media unexpectedly becomes unavailable	<audio>, <video>	Here
onended	When media has reached end	<audio>, <video>	Here
onerror	When an error occurs	<audio>, <body>, <embed>, <img>, <object>, <script>, <style>, <video>, EventSource objects	Here
onfocus	When element gets focus	All visible elements	Here
onfocusin	When element is about to get focus (similar to onfocus except also bubbles)	All visible elements	Here
onfocusout	When element is about to lose focus (similar to onblur except also bubbles)	All visible elements	Here
onhashchange	When there has been changes to the anchor part of the URL	<body>	Here
oninput	When element gets user input	All visible elements	Here
oninvalid	When element is invalid	All visible elements	Here
onkeydown	When user is pressing key	All visible elements	Here
onkeypress	When user presses a key	All visible elements	Here
onkeyup	When user releases a key	All visible elements	Here
onload	When element finishes loading	<body>, <iframe>, <img>, <input>, <link>, <script>, <style>	Here
onloadeddata	When media data is loaded	<audio>, <video>	Here
onloadedmetadata	When metadata (dimensions, duration, …) loaded	<audio>, <video>	Here
onloadstart	Just before loading starts	<audio>, <video>	Here
onmessage	When message is triggered	For handling errors	Here
onmousedown	When mouse button is pressed down on an element	All visible elements	Here
onmouseenter	When mouse pointer moves over an element	All visible elements	Here
onmouseleave	When mouse pointer moves out of an element	All visible elements	Here
onmousemove	For as long as mouse pointer is moving over an element	All visible elements	Here

onmouseout	When mouse pointer moves out of an element	All visible elements	Here
onmouseover	When mouse pointer moves over an element	All visible elements	Here
onmouseup	When mouse button is released over an element	All visible elements	Here
onmousewheel	When mouse wheel is being scrolled over an element (depreciated: use onwheel instead)	All visible elements	Here
onoffline	When browser starts to work offline	<body>	Here
ononline	When browser starts to work online	<body>	Here
onopen	When a connection to an event source is opened	An EventSource object	Here
onpagehide	When user navigates away from a page	<body>	Here
onpageshow	When user navigates to a page	<body>	Here
onpaste	When user pastes content in an element	All visible elements	Here
onpause	When media is paused	<audio>, <video>	Here
onplay	When media is ready to start playing	<audio>, <video>	Here
onplaying	When media has actually started playing	<audio>, <video>	Here
onpopstate	When window's history changes	<body>	Here
onprogress	When browser is in process of getting media data	<audio>, <video>	Here
onratechange	When playback rate changes (e.g. user switches to fast forward)	<audio>, <video>	Here
onreset	When reset button in a form is clicked	<form>	Here
onresize	When browser window is being resized	<body>	Here
onscroll	When element's scrollbar is being scrolled	All visible elements	Here
onsearch	When user writes something in search field (for an <input> element of type = search)	<input>	Here
onseeked	When seeking attribute is set to false (i.e. seeking finished)	<audio>, <video>	Here
onseeking	When seeking attribute is set to true (i.e. seeking is active)	<audio>, <video>	Here
onselect	When element gets selected	All visible elements	Here
onshow	When <menu> element is	<menu>	Here

	shown as a context menu		
onstalled	When browser is unable to fetch media data (for whatever reason)	<audio>, <video>	Here
onstorage	When web storage area is updated	<body>	Here
onsubmit	When a form is submitted	<form>	Here
onsuspend	When fetching media data is stopped before completely loaded (for whatever reason)	<audio>, <video>	Here
ontimeupdate	When playing position has changed (e.g. user fast forwards to new position in media)	<audio>, <video>	Here
ontoggle	When user opens or closes <details> element	<details>	Here
ontouchcancel	When touch is interrupted	Touch-sensitive elements	Here
ontouchend	When finger is removed from touch screen	Touch-sensitive elements	Here
ontouchmove	When finger is dragged across touch screen	Touch-sensitive elements	Here
ontouchstart	When finger is placed on touch screen	Touch-sensitive elements	Here
onunload	When page has unloaded (or browser window closed)	<body>	Here
onvolumechange	When volume changed (or muted)	<audio>, <video>	Here
onwaiting	When media has paused but is expected to resume (e.g. media has paused to buffer more data)	<audio>, <video>	Here
onwheel	When mouse wheel rolls up or down over an element	All visible elements	Here
transitionend	When CSS transition ends	Any element with a CSS transition	Here

Most of these event attributes are new in HTML 5. The ones that are not are: onabort, onblur, onchange, onclick, oncopy, oncut, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onload, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onmousewheel, onpaste, onsearch, onselect, onsubmit and onunload.

Global attributes

[HTMLGlobalAttributes]

Different HTML elements can have attributes that specify how they should be formatted or interpreted or allow further characterisation of the element, which can be either standard attributes or event attributes. Some standard attributes can apply to essentially all HTML elements. These 'global' attributes include:

Attribute	Description	Applicable to	More
`accesskey`	Shortcut key to activate/focus element	All	Here
`class`	One or more class names for the element	All	Here. Refers to a class in a CSS style
`contenteditable`	Indicates whether content is editable	All	Here. New in HTML 5
`contextmenu`	Specifies context menu (i.e. what appears when right-click)	All	Here. New in HTML 5
`data-*`	Custom data private to page of application	All	Here. New in HTML 5
`dir`	Text direction for element content	All	Here. New in HTML 5
`draggable`	Whether element is draggable	All	Here. New in HTML 5
`dropzone`	Whether dragged data is copied, moved or linked when dropped	All	Here. New in HTML 5
`hidden`	Whether element is not relevant	All	Here. New in HTML 5
`id`	Unique id for an element (for e.g. JavaScript)	All	Here
`lang`	Language of an element's content	All	Here
`spellcheck`	Indicates whether element is to have its spelling and grammar checked	All	Here. New in HTML 5
`style`	Inline CSS style for an element	All	Here
`tabindex`	Tab order of an element	All	Here
`title`	Extra information about element	All	Here
`translate`	Whether content of an element should be translated	All	Here. New in HTML 5

Individual HTML Attributes:

accept

[HTMLAttributeAccept]

The HTML `accept` attribute specifies the types of file accepted by the server. It applies to <input> elements but only if type = `file` (and, prior to HTML 5, to <area> elements).

Valid attribute values (when used with <area> elements) include:

Value	Description

file_type	Not supported in HTML 5. Alternative text to display

Valid attribute values (when used with <input> elements) include:

Value	Description
file_extension	A file extension starting with a full stop, e.g. `.png`, `.jpg`, `.pdf`, `.doc`
media_type	A valid media type, see e.g. http://www.iana.org/assignments/media-types/media-types.xhtml
`audio/*`	Indicates any audio (sound) file is acceptable
`image/*`	Indicates any image file is acceptable
`video/*`	Indicates any video file is acceptable

More than one value can applied, if separated by commas.

accept-charset

[Nematrian website page: HTMLAttributeAcceptCharset, © Nematrian 2017]

The HTML `accept-charset` attribute specifies the character encodings used for submission of a <form> element.

Valid attribute values (when used with <form> elements) include:

Value	Description
character_set	Character encodings to be used when submitting the form

Common *character_sets* include:

- `UTF-8`: Unicode
- `ISO-8859-1`: character encoding for Latin alphabet

Multiple character_sets are acceptable and in HTML 5 need to be delimited (separated) by spaces. The default value is the reserved string `UNKNOWN`, which indicates that the encoding is the same as that for the document containing the <form> element.

accesskey

[HTMLAttributeAccesskey]

The HTML `accesskey` attribute indicates the shortcut key used to activate / focus an element. In HTML 5 it can in principle be used with any element, although in practice it may not be of much use with some elements. Different browsers use different ways of accessing shortcut keys (sometimes using the Alt key (or Alt and Shift keys simultaneously) or the Control key, in combination with a specified character.

Valid attribute values (when used with <form> elements) include:

Value	Description
character	The shortcut key character used to activate / focus the element

action

[HTMLAttributeAction]

The HTML `action` attribute indicates where to send form-data for a <form> element when the form is submitted.

Valid attribute values (when used with a <form> element) include:

Value	Description
URL	Where to send the form-data when form is submitted

align

[HTMLAttributeAlign]

The HTML `align` attribute indicates the alignment of the element versus surrounding elements. It is not supported in HTML 5 (instead use CSS, e.g. `<div style="text-align:center"> … </div>`).

alt

[HTMLAttributeAlt]

The HTML `alt` attribute indicates the alternative text to show when original content (e.g. an image) fails to display. It applies to <area>, <img> and <input> elements.

There are several possible reasons why an image might not display, e.g. there might be a slow connection, the content location might be wrongly specified or the user might be using a screen reader because he or she is partly sighted). Some old browsers showed the value of the `alt` attribute as a tooltip, but modern browsers use the title attribute instead for this purpose.

Valid attribute values (when used with <area>, <img> and <input> elements) include:

Value	Description
text	Alternative text to display

aminationend

[HTMLAttributeAnimationend]

The HTML `aminationend` attribute specifies the event that is triggered when a CSS animation ends. It applies to HTML elements that have CSS animatable elements. It seems to be necessary to set it using JavaScript.

aminationiteration

[HTMLAttributeAnimationiteration]

The HTML `aminationiteration` attribute specifies the event that is triggered when a CSS animation is repeated. It applies to HTML elements that have CSS animatable elements. It seems to be necessary to set it using JavaScript.

aminationstart

[HTMLAttributeAnimationstart]

The HTML `aminationstart` attribute specifies the event that is triggered when a CSS animation starts. It applies to HTML elements that have CSS animatable elements. It seems to be necessary to set it using JavaScript.

async

[HTMLAttributeAsync]

The HTML `async` attribute indicates if a script is to be executed asyncronously. It applies to <script> elements. It only in practice applies to external scripts, so should only be used if the `src` attribute is also present.

The `async` and defer attributes work in tandem as follows:

- If `async` is present then the (external) script is executed asynchronously with the rest of the page (with the script being executed while the page continues to be parsed)
- If `async` is not present but `defer` is present then the (external) script is executed when the page has finished parsing
- If neither `async` or `defer` is present then the (external) script is fetched and executed immediately, before further parsing of the page

Valid attribute values (when used with <script> elements) include:

Value	Description
`async`	Script should be executed asyncronously

autocomplete

[HTMLAttributeAutocomplete]

The HTML `autocomplete` attribute indicates whether an element has autocomplete capability enabled. This enables the browser to display options to fill in the field, based on previously typed characters. It applies to <form> and <input> elements (if the <input> element is type: `text`, `search`, `url`, `tel`, `email`, `password`, `datepickers`, `range` or `color`). Sometimes an autocomplete function needs to be enabled within the browser for autocomplete to work.

Valid attribute values (when used with <form> and <input> elements) include:

Value	Description
`on`	Form should have autocomplete on
`off`	Form should have autocomplete off

autofocus

[HTMLAttributeAutofocus]

The HTML `autofocus` attribute indicates whether an element should automatically get focus when the page loads. It applies to <button>, <input>, <keygen>, <select> and <textarea> elements.

Valid attribute values (when used with <button>, <input>, <keygen>, <select> and <textarea> elements) include:

Value	Description
`autofocus`	Element should automatically get focus when page loads

autoplay

[HTMLAttributeAutoplay]

The HTML `autoplay` attribute indicates whether an audio or video should start playing as soon as it is ready. It applies to <audio> and <video> elements.

Valid attribute values (when used with an <audio> and <video> element) include:

Value	Description
`autoplay`	Media to start playing as soon as ready

bgcolor

[HTMLAttributeBgcolor]

The HTML `bgcolor` attribute indicates the background colour of an element. It is no longer supported in HTML 5 (instead use CSS, e.g. `<div style="background-color:yellow">…</div>`).

border

[HTMLAttributeBorder]

The HTML `border` attribute indicates the width of the border of an element. It is no longer supported in HTML 5 (instead use CSS).

challenge

[HTMLAttributeChallenge]

The HTML `challenge` attribute indicates that the value of an element should be challenged when submitted. It applies to <keygen> elements. Note: it appears likely that <keygen> elements will be dropped from future versions of HTML so it may be desirable not to use <keygen> elements.

Valid attribute values (when used with <keygen> elements) include:

Value	Description
`challenge`	Value of element should be challenged when submitted

charset

[HTMLAttributeCharset]

The HTML `charset` attribute specifies the character encoding to use. It applies to <meta> and <script> elements.

Common values for this attribute include:

- UTF-8: the character encoding for Unicode
- ISO-8859-1: the character encoding for the Latin alphabet

It can be overridden for a specific element by setting the lang attribute of that element. The `charset` attribute is new in HTML 5 and replaces the need to set the content type via HTML such as: `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">` (although using the http-equiv approach is still allowed).

Valid attribute values (when used with <meta> and <script> elements) include:

Value	Description
character_set	The character encoding for the document

checked

[HTMLAttributeChecked]

The HTML `checked` attribute specifies that the (sub)-element should be pre-selected (i.e. 'checked') when the page first loads. It applies to <input> elements (if type = `checkbox` or type = `radio`). It also applies to <menuitem> elements (but these are not currently supported by many browsers)

Valid attribute values (when used with an <input> element) include:

Value	Description
`checked`	Sub-element should be pre-selected

Valid attribute values (when used with a <menuitem> element) include:

Value	Description
`checked`	Indicates that command/menu item should be checked when page loads. Only applies to type = `radio` or type = `checkbox`

cite

[HTMLAttributeCite]

The HTML `cite` attribute provides a URL which explains a quote, deleted text or inserted text. It applies to <blockquote>, <del>, <ins> and <q> elements. It has no visual effect in typical web browsers but can be used by screen readers.

Valid attribute values (when used with <blockquote>, <del>, <ins> elements) include:

Value	Description
URL	Source of quote, deletion or insertion

class

[HTMLAttributeClass]

The HTML `class` attribute indicates one or more style class names (as per CSS) that apply to the element. If more than one class is to be applied to the same element then they should be separated by a space.

Valid attribute values include:

Value	Description
CSSclass	CSS style class

color

[HTMLAttributeColor]

The HTML `color` attribute indicates the colour of the text of an element. It is no longer supported in HTML 5 (instead use CSS).

cols

[HTMLAttributeCols]

The HTML `cols` attribute indicates the visible width of a <textarea> element (in number of characters).

Valid attribute values (when used with <textarea> elements) include:

Value	Description
integer	Visible width (in characters) of text area

The visible height of a <textarea> element can be set using the rows attribute. The size of a <textarea> element can also be set using CSS height and width properties.

colspan

[HTMLAttributeColspan]

The HTML `colspan` attribute indicates the number of columns a table cell should span. It applies to <td> and <th> elements.

A value of zero, i.e. using colspan="0", in theory has a special meaning, namely that the cell should be spanned to the last column of the column group, but this is not recognised by some browsers.

Valid attribute values (when used with <td> and <th> element) include:

Value	Description
integer	Number of columns a cell should span

content

[HTMLAttributeContent]

The HTML `content` attribute indicates the value associated with the http-equiv or name attribute within a <meta> element.

Valid attribute values (for <meta>) include:

Value	Description
text	Value associated with the relevant http-equiv or name attribute

contenteditable

[HTMLAttributeContenteditable]

The HTML `contenteditable` attribute indicates whether content of an element is editable.

Valid attribute values include:

Value	Description
`contenteditable`	Content of element is editable

contextmenu

[HTMLAttributeContextmenu]

The HTML `contextmenu` attribute indicates the context menu (i.e. what appears when the mouse is right-clicked). At the time of writing (early 2018) it did not appear to be supported by many browsers.

Valid attribute values include:

Value	Description
text	Text to display when mouse is right-clicked

controls

[HTMLAttributeControls]

The HTML `controls` attribute indicates whether <audio> and <video> controls (such as play and pause buttons) should be displayed.

Valid attribute values (when used with <audio> and <video> elements) include:

Value	Description
`controls`	Controls should be displayed (e.g. play / pause button, fast forward etc.

coords

[HTMLAttributeCoords]

The HTML `coords` attribute indicates the coordinates of an <area>. It, together with the shape attribute specify the size, shape and position of the area.

Valid attribute values (for <area>) include:

Value	Description
x1, *y1*, *x2*, *y2*	Coordinates of the left, top (*x1*, *y1*) and right, bottom (*x2*, *y2*) corners of a rectangle, if `shape = "rect"`
x, *y*, *r*	Coordinates of the circle centre (*x*, *y*) and circle radius (*r*), if `shape = "circle"`
x1, *y1*, *x2*, *y2*,..., *xn*, *yn*	Coordinates of corners of polygon. If the first and last coordinate pairs are not the same then the browser will add another coordinate pair to complete the polygon, if `shape = "poly"`

crossorigin

[HTMLAttributeCrossorigin]

The HTML `crossorigin` attribute indicates how the element handles cross-origin requests. It can apply to <img> and <link> elements (and some other elements for some browsers).

Valid attribute values (when used with <img> and <link> elements) include:

Value	Description
	(default), i.e. blank. CORS will not be used at all.
`anonymous`	CORS requests for element will not have credentials flag set, i.e. no user credentials will be exchanged via e.g. cookies, client-side SSL certificates or HTTP authentication
`use-credentials`	CORS requests for element will have credentials flag set, i.e. request will provide credentials

data

[HTMLAttributeData]

The HTML `data` attribute specifies the URL of a resource to be used by an <object> element.

Valid attribute values (when used with <object> elements) include:

Value	Description
URL	URL of resource used by element

data-*

[HTMLAttributeDataCustom]

The HTML `data-*` attribute provides a means of storing custom data specific to a page. It can be applied to all HTML elements and can be accessed by JavaScript embedded within the page.

The `data-*` attributes have two components:

(a) The name, which is the * part of the overall attribute name, which should not contain any uppercase letters
(b) The attribute value, which can be any string

E.g.

```
<ul>
  <li data-continent="Europe">Spain</li>
  <li data-continent="Asia">Japan</li>
</ul>
```

datetime

[HTMLAttributeDatetime]

The HTML `datetime` attribute specifies the date and time of a <del>, <ins> or <time> element.

Valid attribute values (when used with <del>, <ins> elements) include:

Value	Description
YYYY-MM-DDThh:mm:ssTZD	A date

Valid attribute values (when used with <time> elements) include:

Value	Description
datetime	A machine-readable date/time for the <time> element

default

[HTMLAttributeDefault]

The HTML `default` attribute specifies the default <menuitem> or <track> element that will be enabled unless user preferences specify otherwise.

Valid attribute values (when used with <menuitem> and <track> elements) include:

Value	Description
`default`	Marks relevant element as default

defer

[HTMLAttributeDefer]

The HTML `defer` attribute specifies that JavaScript within (or refenced by) a <script> should only be executed once the page has finished being parsed by the browser. It only in practice applies to external scripts.

The async and `defer` attributes work in tandem as follows:

- If `async` is present then the (external) script is executed asynchronously with the rest of the page (with the script being executed while the page continues to be parsed)
- If `async` is not present but `defer` is present then the (external) script is executed when the page has finished parsing
- If neither `async` or `defer` is present then the (external) script is fetched and executed immediately, before further parsing of the page

Valid attribute values (when used with <script> elements) include:

Value	Description
`defer`	Script should be deferred until page has finished being parsed by the browser

dir

[HTMLAttributeDir]

The HTML `dir` attribute specifies the direction of the text content of an element.

Valid attribute values (for <bdo>) include:

Value	Description
`ltr`	Left-to-right
`rtl`	Right-to-left

dirname

[HTMLAttributeDirname]

The HTML `dirname` attribute indicates that the text direction for the content of an element will be submitted. It applies to <input> and <textarea> elements. Currently, not all major browsers support this attribute.

The value of the `dirname` attribute is always the name of the input field, followed by `.dir`, i.e. valid attribute values (when used with <input> and <textarea> elements) are:

Value	Description

inputfieldname`.dir`	Indicates that the text direction of the input field with name *inputfieldname* will be specified

disabled

[HTMLAttributeDisabled]

The HTML `disabled` attribute indicates that the element or group of elements should be disabled. It applies to <button>, <fieldset>, <input>, <keygen>, <menuitem>, <optgroup>, <option>, <select> and <textarea> elements.

Valid attribute values (when used with <button>, <fieldset>, <input>, <keygen>, <menuitem>, <optgroup>, <option>, <select> and <textarea> elements) include:

Value	Description
`disabled`	Element (button, group of related form elements etc.) should be disabled

download

[HTMLAttributeDownload]

The HTML `download` attribute indicates that the target resource will be downloaded when the user clicks on the hyperlink. It applies to <a> and <area> elements.

Valid attribute values (when used with <a> and <area> elements) include:

Value	Description
filename	Resource to be downloaded

draggable

[HTMLAttributeDraggable]

The HTML `draggable` attribute indicates whether an element is draggable.

Valid attribute values include:

Value	Description
`draggable`	Element is draggable

dropzone

[HTMLAttributeDropzone]

The HTML `dropzone` attribute indicates whether dragged material is copied, moved or linked to when it is dropped. It does not seem currently to be supported by many browsers

Valid attribute values include:

Value	Description

`copy`	Dropping will create a copy of the element that was dragged
`move`	Dropping will result in the element that was dragged being moved to this new location
`link`	Dropping will create a link to the dragged data

enctype

[HTMLAttributeEnctype]

The HTML `enctype` attribute indicates how form-data should be encoded when submitted to the server. It applies to <form> elements and then only for method = `post`.

Valid attribute values (for <form>) include:

Value	Description
`application/x-www-form-urlencoded`	(Default). Input control names and values are URL encoded. Each name value pair is separated by a '`&`' and the name is separated from the value by '=' (as per a usual HTML query string)
`multipart/form-data`	Used for submitting forms that contain files, binary data and other non-ASCII data
`text/plain`	Obsolete. No encoding is applied (is only retained for old browser compatibility)

for

[HTMLAttributeFor]

The HTML `for` attribute indicates which form element(s) a label calculation is linked to. It applies to <label> and <output> elements.

Valid attribute values (when used with <label> elements) include:

Value	Description
elementID	Indicates to which form element the <label> belongs

Valid attribute values (when used with <output> elements) include:

Value	Description
elementID	Indicates relationship between calculation result and elements used calculation

form

[HTMLAttributeForm]

The HTML `form` attribute indicates the name of the form to which the element belongs. It applies to <button>, <fieldset>, <input>, <keygen>, <label>, <meter>, <object>, <output>, <select> and <textarea> elements.

Valid attribute values (when used with <button>, <fieldset>, <input>, <keygen>, <label>, <meter>, <object>, <output>, <select> and <textarea> elements) include:

Value	Description
formID	One or more forms to which element belongs

formaction

[HTMLAttributeFormaction]

The HTML `formaction` attribute indicates where to send form-data to when a form is submitted. It applies to <button> and <input> elements and then only for type = `submit`.

Valid attribute values (when used with <button> and <input> elements) include:

Value	Description
URL	Where form-data is sent to when a form is submitted. For <button> elements it only applies if the button type = `"submit"`

formenctype

[HTMLAttributeFormenctype]

The HTML `formenctype` attribute indicates how form-data should be encoded before sending it to a server. It applies to <button> and <input> elements and then only for type = `submit` or type = `image`. It overrides the enctype attribute of the <form> element containing the element.

Valid attribute values (when used with <button> and <input> elements include:

Value	Description
`application/x-www-form-urlencoded`	(Default). All characters are URL encoded before being sent (with spaces converted to + characters and special characters converted to ASCII Hex values
`multipart/form-data`	No characters encoded
`text/plain`	Spaces converted to + characters but no conversion applied to (other) special characters

formmethod

[HTMLAttributeFormmethod]

The HTML `formmethod` attribute indicates how to send form-data (i.e. which HTTP method to use). It applies to <button> and <input> elements and then only for type = `submit`.

Valid attribute values (when used with <button> and <input> elements) include:

Value	Description
HTTPmethod	HTTP method (get or post). See here for more details

formnovalidate

[HTMLAttributeFormnovalidate]

The HTML `formnovalidate` attribute indicates that form-data should not be validated prior to submission to server. It applies to <button> and <input> elements and then only for type = `submit`.

Valid attribute values (when used with <button> and <input> elements) include:

Value	Description
`formnovalidate`	Do not validate form

formtarget

[HTMLAttributeFormtarget]

The HTML `formtarget` attribute indicates where to display the response that is received after submitting form. It applies to <button> and <input> elements and then only for type = `submit`.

Valid attribute values (when used with <button> and <input> elements) include:

Value	Description
`_blank`	Opens linked document in a new window or tab
`_self`	Opens linked document in parent frame
`_parent`	(default value). Opens linked document in the same window or tab as was clicked
`_top`	Opens linked document in full body of the window
framename	Opens linked document in named frame

headers

[HTMLAttributeHeaders]

The HTML `headers` attribute identifies one or more header cells that a specific cell is related to. It applies to <td> and <th> elements.

Valid attribute values (when used with <td> and <th> element) include:

Value	Description
header_id	One or more header cells a cell is related to

height

[HTMLAttributeHeight]

The HTML `height` attribute indicates the height of an element. It applies to <canvas>, <embed>, <iframe>, <img>, <input>, <object> and <video> elements.

Valid attribute values (when used with <canvas>, <embed>, <iframe>, <img>, <input>, <object> and <video> elements) include:

Value	Description
number	Width of element or embedded content in pixels, e.g. `width="20"`
percentage	Width as a percentage of surrounding element, e.g. `width="30%"`

hidden

[HTMLAttributeHidden]

The HTML `hidden` attribute indicates whether an element is hidden.

Valid attribute values include:

Value	Description
`hidden`	Element is hidden

high

[HTMLAttributeHigh]

The HTML `high` attribute indicates a range that is considered to constitute a high value for a <meter> element.

Valid attribute values (when used with <meter> elements) include:

Value	Description
number	(Floating point) number defining number above which value is deemed 'high'. Should be lower than the max attribute and higher than the low attribute

href

[HTMLAttributeHref]

The HTML `href` attribute indicates the URL of the page that link goes to (or for the <base> element the URL that forms the base for relative URLS). It applies to <a>, <area>, <base> and <link> elements.

Valid attribute values (when used with <a>, <area>, <base>, <link> elements) include:

Value	Description
URL	URL location of linked document

If an element has an `href` attribute then the corresponding DOM object usually supports the following additional properties which can be thought of as variants of the `href` attribute:

Value	Description
`hash`	Anchor part of `href` attribute
`host`	Hostname and port part of `href` attribute
`hostname`	Hostname part of `href` attribute
`origin`	Returns protocol, hostname and port part of `href` attribute

`password`	Password part of `href` attribute
`pathname`	Pathname part of `href` attribute
`port`	Port part of `href` attribute
`protocol`	Protocol part of `href` attribute
`search`	Querystring part of `href` attribute
`username`	Username part of `href` attribute

hreflang

[HTMLAttributeHreflang]

The HTML `hreflang` attribute indicates the language of the linked document. It applies to <a>, <area> and <link> elements.

Valid attribute values (when used with <a>, <area>, <link> elements) include:

Value	Description
language-code	Language of text in linked document

http-equiv

[HTMLAttributeHttpEquiv]

The HTML `http-equiv` attribute provides the HTTP header for the information / value of an attribute within a <meta> element.

Valid attribute values (when used with <meta>) include:

Value	Description

`content-type`	The character encoding for the document, e.g.:
	`<meta http-equiv="content-type" content="text/html; charset=UTF-8">`
	Note that using HTML 5 it is now possible to set the character set more directly, using e.g.:
	`<meta charset="UTF-8">`

`default-style`	The preferred style sheet to use for the page, e.g.: `<meta http-equiv="default-style" content="`*preferred stylesheet*`">` The value of the relevant content attribute must either match the value of the title attribute of a link element (linked to an external style sheet) or a style element in the same document
`refresh`	The time interval for the document to refresh itself, e.g. `<meta http-equiv="refresh" content="60">` It is recommended that this option is used sparingly, since it takes control of the page away from the user. Often better will be to achieve a similar effect using JavaScript

icon

[HTMLAttributeIcon]

The HTML `icon` attribute indicates the icon that should be used for a <menuitem> element.

Valid attribute values (when used with <menuitem> elements) include:

Value	Description
URL	Location of icon

id

[HTMLAttributeId]

The HTML `id` attribute indicates the unique id (identifier) for an element.

Valid attribute values include:

Value	Description
text	Unique id (identifier)

ismap

[HTMLAttributeIsmap]

The HTML `ismap` attribute indicates if an <img> element is a server-side image-map.

Valid attribute values (when used with <img> elements) include:

Value	Description
`ismap`	Specifies whether the <img> element is part of a server-side image-map, i.e. has clickable areas. The click coordinates are then sent to the server as part of the URL query string. It is only allowed if the <img> element is a descendant of an <a> element with a valid `href` attribute.

keytype

[HTMLAttributeKeytype]

The HTML `keytype` attribute specifies the security algorithm of a key. It applies to <keygen> elements.

Valid attribute values (when used with <keygen> elements) include:

Value	Description
`rsa`	(Default). Use RSA security algorithm (user may be given choice of key strength)
`dsa`	Use DSA security algorithm (user may be given choice of key strength)
`ec`	Use EC security algorithm (user may be given choice of key strength)

kind

[HTMLAttributeKind]

The HTML `kind` attribute specifies the kind of a text track (e.g. whether a subtitle). It applies to <track> elements.

Valid attribute values (when used with <track> elements) include:

Value	Description
`captions`	Track relates to translation of dialogue
`chapters`	Track relates to chapter titles (helps when navigating the media resource)
`descriptions`	Track relates to text description of video content
`metadata`	Track defines content used by scripts
`subtitles`	Track defines subtitles

label

[HTMLAttributeLabel]

The HTML `label` attribute specifies the title of the track or group (for <optgroup>, <option> and <track> elements) or the visible label to give to a menu (for <menu> elements).

Valid attribute values (when used with <menu>, <optgroup>, <option> and <track>elements) include:

Value	Description
text	Visible label

lang

[HTMLAttributeLang]

The HTML `lang` attribute specifies the language of an element's content.

Valid attribute values include:

Value	Description
language-code	Language of content

list

[HTMLAttributeList]

The HTML `list` attribute specifies the <datalist> element that contains pre-defined options for an <input> element.

Valid attribute values (when used with <input> elements) include:

Value	Description
datalist_id	ID of relevant <datalist> element

loop

[HTMLAttributeLoop]

The HTML `loop` attribute specifies whether an <audio> or <video> element is to start over again when it finishes.

Valid attribute values (when used with <audio> and <video> elements) include:

Value	Description
`loop`	Media to start over again each time it finishes

low

[HTMLAttributeLow]

The HTML `low` attribute indicates a range that is considered to constitute a low value for a <meter> element.

Valid attribute values (when used with <meter> elements) include:

Value	Description
number	(Floating point) number defining number below which value is deemed 'low'. Should be higher than the min attribute and lower than the high attribute

manifest

[HTMLAttributeManifest]

The HTML `manifest` attribute specifies the address of the document's cache manifest (for offline browsing). It applies to <html> elements.

Valid attribute values (when used with <html> elements) include:

Value	Description
URL	URL of document's cache manifest (facilitates offline browsing)

max

[HTMLAttributeMax]

The HTML `max` attribute specifies the maximum value applicable to an <input>, <meter> or <progress> element.

Valid attribute values (when used with <input>, <meter> elements) include:

Value	Description
number	Maximum (numerical) value for element
date	Maximum (date) value for an <input> element

maxlength

[HTMLAttributeMaxlength]

The HTML `maxlength` attribute specifies the maximum number of characters allowed in an <input> or <textarea> element.

Valid attribute values (when used with <input> and <textarea> elements) include:

Value	Description
integer	Maximum number of characters allowed to be inputted using element

media

[HTMLAttributeMedia]

The HTML `media` attribute specifies the media or device that the linked document is optimised for. It applies to <a>, <area>, <link>, <source> and <style> elements.

Valid attribute values (when used with <a>, <area>, <link>, <source> and <style> elements) include:

Value	Description
media-query	Indicates the media or device type the target URL is optimised for

method

[HTMLAttributeMethod]

The HTML `method` attribute specifies the HTTP method used when sending form-data. It applies to <form> elements.

Valid attribute values (for <form>) include:

Value	Description
HTTPmethod	HTTP method (get or post). See here for more details

min

[HTMLAttributeMin]

The HTML `min` attribute specifies the minimum value applicable to an <input> or <meter> element.

Valid attribute values (when used with <input>, <meter> elements) include:

Value	Description
number	Minimum (numerical) value for element
date	Minimum (date) value for an <input> element

multiple

[HTMLAttributeMultiple]

The HTML `multiple` attribute indicates that a user can enter more than one value into an <input> or <select> element.

Valid attribute values (when used with <input> and <select> elements) include:

Value	Description
`multiple`	Indicates that more than one value can be entered into element

muted

[HTMLAttributeMuted]

The HTML `muted` attribute indicates whether the audio output of an <audio> or <video> element should be muted.

Valid attribute values (when used with <audio> and <video> elements) include:

Value	Description
`muted`	Audio output should be muted

name

[HTMLAttributeName]

The HTML `name` attribute generally specifies the name of an element. It applies to <button>, <fieldset>, <form>, <iframe>, <input>, <keygen>, <map>, <meta>, <object>, <output>, <param>, <select> and <textarea> elements.

Valid attribute values (when used with <button>, <fieldset>, <form>, <iframe>, <input>, <keygen>, <object>, <output>, <select> and <textarea> elements) include:

Value	Description
name	Name for element or associated element

Valid attribute values (when used with <map> elements) include:

Value	Description
name	Name associated with the <img> element's usemap attribute that creates a relationship between the image and the map

Valid attribute values (when used with <meta> elements) include:

Value	Description
`application-name`	Name of web application to which page is associated
`author`	Author of document
`description`	Description of page (often picked up by search engines to show with

	results of searches)
`generator`	One or more software packages that have generated the document
`keywords`	A comma-separated list of keywords relevant to the page (again helpful for search engines). Specifying this piece of metadata helps with search engine optimisation
`viewport`	Information about the viewport, i.e. the window in which the user sees the webpage. For example, it is common to include the following element in webpages to improve their viewability across different devices: `<meta name="viewport" content="width=device-width, initial-scale=1.0">` The `width=device-width` part of the content attribute indicates that the width of the page should adapt to the screen width, and the `initial-scale=1.0` part identifies the initial zoom level used when the page is first loaded into the browser.

novalidate

[HTMLAttributeNovalidate]

The HTML `novalidate` attribute indicates whether a <form> element should not be validated when submitted.

Valid attribute values (for <form>) include:

Value	Description
`novalidate`	Whether form-data (i.e. input) should not be validated when submitted

onabort

[HTMLAttributeOnabort]

The HTML `onabort` attribute specifies the event that is triggered if the document is aborted. It applies to <audio>, <embed>, <img>, <object> and <video> elements.

onafterprint

[HTMLAttributeOnafterprint]

The HTML `onafterprint` attribute specifies the event that is triggered after a document is printed. It applies to <body> elements.

onbeforeprint

[HTMLAttributeOnbeforeprint]

The HTML `onbeforeprint` attribute specifies the event that is triggered before a document is printed. It applies to <body> elements.

onbeforeunload

[HTMLAttributeOnbeforeunload]

The HTML `onbeforeunload` attribute specifies the event that is triggered just before a document is unloaded. It applies to <body> elements.

It can be used to return a message if the user is just about to leave the page.

onblur

[HTMLAttributeOnblur]

The HTML `onblur` attribute specifies the event that is triggered when an element loses focus. It applies to all visible elements.

oncanplay

[HTMLAttributeOncanplay]

The HTML `oncanplay` attribute specifies the event that is triggered when a file is ready to start playing (i.e. when it has buffered enough to begin). It applies to <audio>, <embed>, <object> and <video> elements.

oncanplaythrough

[HTMLAttributeOncanplaythrough]

The HTML `oncanplaythrough` attribute specifies the event that is triggered when a file is ready to play all the way to its end without pausing for buffering. It applies to <audio> and <video> elements.

onchange

[HTMLAttributeOnchange]

The HTML `onchange` attribute specifies the event that is triggered when an element's value changes. It applies to all visible elements.

onclick

[HTMLAttributeOnclick]

The HTML `onclick` attribute specifies the event that is triggered when an element is clicked (mouse clicked). It applies to all visible elements.

oncontextmenu

[HTMLAttributeOncontextmenu]

The HTML `oncontextmenu` attribute specifies the event that is triggered when a context menu is triggered. It applies to all visible elements.

oncopy

[HTMLAttributeOncopy]

The HTML `oncopy` attribute specifies the event that is triggered when the content of an element is copied. It applies to all visible elements.

oncuechange

[HTMLAttributeOncuechange]

The HTML `oncuechange` attribute specifies the event that is triggered when the cue changes in a <track> element. It applies to <track> elements.

oncut

[HTMLAttributeOncut]

The HTML `oncut` attribute specifies the event that is triggered when the content of an element is cut. It applies to all visible elements.

ondblclick

[HTMLAttributeOndblclick]

The HTML `ondblclick` attribute specifies the event that is triggered when an element is double-clicked (mouse double-clicked). It applies to all visible elements.

ondrag

[HTMLAttributeOndrag]

The HTML `ondrag` attribute specifies the event that is triggered when an element is dragged. It applies to all visible elements.

ondragend

[HTMLAttributeOndragend]

The HTML `ondragend` attribute specifies the event that is triggered at the end of a drag operation. It applies to all visible elements.

ondragenter

[HTMLAttributeOndragenter]

The HTML `ondragenter` attribute specifies the event that is triggered when an element has been dragged to a valid drop target. It applies to all visible elements.

ondragleave

[HTMLAttributeOndragleave]

The HTML `ondragleave` attribute specifies the event that is triggered when an element is dragged outside a valid drop target. It applies to all visible elements.

ondragover

[HTMLAttributeOndragover]

The HTML `ondragover` attribute specifies the event that is triggered when an element is being dragged over a valid drop target. It applies to all visible elements.

ondragstart

[HTMLAttributeOndragstart]

The HTML `ondragstart` attribute specifies the event that is triggered at the start of a drag. It applies to all visible elements.

ondrop

[HTMLAttributeOndrop]

The HTML `ondrop` attribute specifies the event that is triggered when a dragged element is dropped. It applies to all visible elements.

ondurationchange

[HTMLAttributeOndurationchange]

The HTML `ondurationchange` attribute specifies the event that is triggered when the length of a media changes. It applies to <audio> and <video> elements.

onemptied

[HTMLAttributeOnemptied]

The HTML `onemptied` attribute specifies the event that is triggered when a media file unexpected becomes unavailable. It applies to <audio> and <video> elements.

onended

[HTMLAttributeOnended]

The HTML `onended` attribute specifies the event that is triggered when a media reaches its end. It applies to <audio> and <video> elements.

onerror

[HTMLAttributeOnerror]

The HTML `onerror` attribute specifies the event that is triggered when an error occurs. It applies to <audio>, <body>, <embed>, <img>, <object>, <script>, <style> and <video> elements.

onfocus

[HTMLAttributeOnfocus]

The HTML `onfocus` attribute specifies the event that is triggered when an element gets focus. It applies to all visible elements.

onfocusin

[HTMLAttributeOnfocusin]

The HTML `onfocusin` attribute specifies the event that is triggered when an element is about to get focus. It is similar to the onfocus attribute except that it also 'bubbles'.

onfocusout

[HTMLAttributeOnfocusout]

The HTML `onfocusout` attribute specifies the event that is triggered when an element is about to lose focus. It is similar to the onblur attribute except that it also 'bubbles'.

onhashchange

[HTMLAttributeOnhashchange]

The HTML `onhashchange` attribute specifies the event that is triggered when there is a change to the anchor part of a URL (i.e. the part after a #). It applies to <body> elements.

oninput

[HTMLAttributeOninput]

The HTML `oninput` attribute specifies the event that is triggered when an element gets user input. It applies to all visible elements.

oninvalid

[HTMLAttributeOninvalid]

The HTML `oninvalid` attribute specifies the event that is triggered when an element is invalid. It applies to all visible elements.

onkeydown

[HTMLAttributeOnkeydown]

The HTML `onkeydown` attribute specifies the event that is triggered when the user is pressing a key. It applies to all visible elements.

onkeypress

[HTMLAttributeOnkeypress]

The HTML `onkeypress` attribute specifies the event that is triggered when the user presses a key. It applies to all visible elements.

onkeyup

[HTMLAttributeOnkeyup]

The HTML `onkeyup` attribute specifies the event that is triggered when the user releases a key. It applies to all visible elements.

onload

[HTMLAttributeOnload]

The HTML `onload` attribute specifies the event that is triggered when an element finishes loading. It applies to <body>, <iframe>, <img>, <input>, <link>, <script> and <style> elements.

onloadeddata

[HTMLAttributeOnloadeddata]

The HTML `onloadeddata` attribute specifies the event that is triggered when data for the current frame is loaded, but not enough data is yet loaded to play the next frame. It applies to <audio> and <video> elements.

onloadedmetadata

[HTMLAttributeOnloadedmetadata]

The HTML `onloadedmetadata` attribute specifies the event that is triggered when metadata (dimensions, duration, …) is loaded. It applies to <audio> and <video> elements.

onloadstart

[HTMLAttributeOnloadstart]

The HTML `onloadstart` attribute specifies the event that is triggered just before loading starts. It applies to <audio> and <video> elements.

During loading the following events occur in the following order:

- onloadstart
- ondurationchange
- onloadedmetadata
- onloadeddata
- onprogress
- oncanplay
- oncanplaythrough

onmessage

[HTMLAttributeOnmessage]

The HTML `onmessage` attribute specifies the event that is triggered when a message is received through an event source.

onmousedown

[HTMLAttributeOnmousedown]

The HTML `onmousedown` attribute specifies the event that is triggered when the mouse button is pressed down on an element. It applies to all visible elements.

onmouseenter

[HTMLAttributeOnmouseenter]

The HTML `onmouseenter` attribute specifies the event that is triggered when the mouse pointer moves onto an element. It applies to all visible elements. It is often used in conjunction with the onmouseleave event.

It is like the onmouseover event (or the onmousemove event), except that the `onmouseenter` event only fires when the mouse first enters the element itself, whereas the onmouseover event also fires in response to the mouse moving into the element from a child element that is located within the original element.

onmouseleave

[HTMLAttributeOnmouseleave]

The HTML `onmouseleave` attribute specifies the event that is triggered when the mouse pointer moves onto an element. It applies to all visible elements. It is often used in conjunction with the onmouseenter event.

It is like the onmouseout event (or the onmousemove event), except that the `onmouseleave` event only fires when the mouse first leaves the element itself, whereas the onmouseout event also fires in response to the mouse moving out of the element into a child element that is located within the original element.

onmousemove

[HTMLAttributeOnmousemove]

The HTML `onmousemove` attribute specifies the event that is triggered for as long as the mouse pointer is moving over an element. It applies to all visible elements.

onmouseout

[HTMLAttributeOnmouseout]

The HTML `onmouseout` attribute specifies the event that is triggered when the mouse pointer moves outside an element. It applies to all visible elements.

onmouseover

[HTMLAttributeOnmouseover]

The HTML `onmouseover` attribute specifies the event that is triggered when the mouse pointer moves over an element. It applies to all visible elements.

onmouseup

[HTMLAttributeOnmouseup]

The HTML `onmouseup` attribute specifies the event that is triggered when the mouse pointer is released over an element. It applies to all visible elements.

onmousewheel

[HTMLAttributeOnmousewheel]

The HTML `onmousewheel` attribute specifies the event that is triggered when the mouse pointer is released over an element. It applies to all visible elements. Depreciated, use onwheel instead.

onoffline

[HTMLAttributeOnoffline]

The HTML `onoffline` attribute specifies the event that is triggered when the browser starts to work offline. It applies to <body> elements.

ononline

[HTMLAttributeOnonline]

The HTML `ononline` attribute specifies the event that is triggered when the browser starts to work online. It applies to <body> elements.

onopen

[HTMLAttributeOnopen]

The HTML `onopen` attribute specifies the event that is triggered when a connection to an event source is opened.

onpagehide

[HTMLAttributeOnpagehide]

The HTML `onpagehide` attribute specifies the event that is triggered when the user navigates away from a page. It applies to <body> elements.

onpageshow

[HTMLAttributeOnpageshow]

The HTML `onpageshow` attribute specifies the event that is triggered when the user navigates to a page. It applies to <body> elements.

onpaste

[HTMLAttributeOnpaste]

The HTML `onpaste` attribute specifies the event that is triggered when the user pastes content in an element. It applies to all visible elements.

onpause

[HTMLAttributeOnpause]

The HTML `onpause` attribute specifies the event that is triggered when a media is paused. It applies to <audio> and <video> elements.

onplay

[HTMLAttributeOnplay]

The HTML `onplay` attribute specifies the event that is triggered when a media is ready to start playing. It applies to <audio> and <video> elements.

onplaying

[HTMLAttributeOnplaying]

The HTML `onplaying` attribute specifies the event that is triggered when a media has started playing. It applies to <audio> and <video> elements.

onpopstate

[HTMLAttributeOnpopstate]

The HTML `onpopstate` attribute specifies the event that is triggered when the window's history changes. It applies to <body> elements.

onprogress

[HTMLAttributeOnprogress]

The HTML `onprogress` attribute specifies the event that is triggered when the browser is in the process of getting media data. It applies to <audio> and <video> elements.

onratechange

[HTMLAttributeOnratechange]

The HTML `onratechange` attribute specifies the event that is triggered when the playback rate of a media changes (e.g. the user switches to fast forward). It applies to <audio> and <video> elements.

onreset

[HTMLAttributeOnreset]

The HTML `onreset` attribute specifies the event that is triggered when the reset button in a form is clicked. It applies to <form> elements.

onresize

[HTMLAttributeOnresize]

The HTML `onresize` attribute specifies the event that is triggered when the browser window is being resized. It applies to <body> elements.

onscroll

[HTMLAttributeOnscroll]

The HTML `onscroll` attribute specifies the event that is triggered when the element's scrollbar is being scrolled. It applies to all visible elements.

onsearch

[HTMLAttributeOnsearch]

The HTML `onsearch` attribute specifies the event that is triggered when user enters something in a search field (for an <input> element of type = `search`). It applies to <input> elements.

onseeked

[HTMLAttributeOnseeked]

The HTML `onseeked` attribute specifies the event that is triggered when the seeking attribute of a media is set to false (i.e. the seeking has finished). It applies to <audio> and <video> elements.

onseeking

[HTMLAttributeOnseeking]

The HTML `onseeking` attribute specifies the event that is triggered when the seeking attribute of a media is set to true (i.e. the seeking is active). It applies to <audio> and <video> elements.

onselect

[HTMLAttributeOnselect]

The HTML `onselect` attribute specifies the event that is triggered when an element gets selected. It applies to all visible elements.

onshow

[HTMLAttributeOnshow]

The HTML `onshow` attribute specifies the event that is triggered when a <menu> element is shown as a context menu. It applies to <menu> elements.

onstalled

[HTMLAttributeOnstalled]

The HTML `onstalled` attribute specifies the event that is triggered when the browser is unable to fetch the media data (for whatever reason). It applies to <audio> and <video> elements.

onstorage

[HTMLAttributeOnstorage]

The HTML `onstorage` attribute specifies the event that is triggered when the web storage area is updated. It applies to <body> elements.

onsubmit

[HTMLAttributeOnsubmit]

The HTML `onsubmit` attribute specifies the event that is triggered when a form is submitted. It applies to <form> elements.

onsuspend

[HTMLAttributeOnsuspend]

The HTML `onsuspend` attribute specifies the event that is triggered when the fetching of media data is stopped before completely loaded (for whatever reason). It applies to <audio> and <video> elements.

ontimeupdate

[HTMLAttributeOntimeupdate]

The HTML `ontimeupdate` attribute specifies the event that is triggered when the playing position in a media has changed (e.g. user fast forwards to a new position in the media). It applies to <audio> and <video> elements.

ontoggle

[HTMLAttributeOntoggle]

The HTML `ontoggle` attribute specifies the event that is triggered when the user opens or closes a <details> element. It applies to <details> elements.

ontouchcancel

[HTMLAttributeOntouchcancel]

The HTML `ontouchcancel` attribute specifies the event that is triggered when touch is interrupted. It applies to touch-sensitive elements.

ontouchend

[HTMLAttributeOntouchend]

The HTML `ontouchend` attribute specifies the event that is triggered when the touching device (usually a finger) is removed from the touch screen. It applies to touch-sensitive elements.

ontouchmove

[HTMLAttributeOntouchmove]

The HTML `ontouchmove` attribute specifies the event that is triggered when the touching device (usually a finger) is dragged across the touch screen. It applies to touch-sensitive elements.

ontouchstart

[HTMLAttributeOntouchstart]

The HTML `ontouchstart` attribute specifies the event that is triggered when the touching device (usually a finger) is placed on the touch screen. It applies to touch-sensitive elements.

onunload

[HTMLAttributeOnunload]

The HTML `onunload` attribute specifies the event that is triggered when the page has unloaded (or the browser window has closed). It applies to <body> elements.

onvolumechange

[HTMLAttributeOnvolumechange]

The HTML `onvolumechange` attribute specifies the event that is triggered when the volume of a media is changed (or muted). It applies to <audio> and <video> elements.

onwaiting

[HTMLAttributeOnwaiting]

The HTML `onwaiting` attribute specifies the event that is triggered when the media has paused but is expected to resume (e.g. the media has paused to buffer more data). It applies to <audio> and <video> elements.

onwheel

[HTMLAttributeOnwheel]

The HTML `onwheel` attribute specifies the event that is triggered when the mouse wheel rolls up or down over an element. It applies to all visible elements. It currently is not supported by all major browsers.

open

[HTMLAttributeOpen]

The HTML `open` attribute indicates whether details in a <details> or <dialog> element should be visible (i.e. open) to the user.

Valid attribute values (for <details>) include:

Value	Description
`open`	Specifies whether the details should be visible to user

Valid attribute values (for <dialog>) include:

Value	Description
`open`	Specifies whether the dialog element is active and hence whether the user can interact with it

optimum

[HTMLAttributeOptimum]

The HTML `optimum` attribute indicates the value that is deemed optimal for the gauge applicable to a <meter> element.

Valid attribute values (when used with <meter> elements) include:

Value	Description
number	(Floating point) number defining optimal value for gauge

pattern

[HTMLAttributePattern]

The HTML `pattern` attribute indicates the format expression that the value of an <input> element is checked against.

Valid attribute values (when used with <input> elements) include:

Value	Description
regular-expression	Regular expression against which input is compared

placeholder

[HTMLAttributePlaceholder]

The HTML `placeholder` attribute indicates the short hint describing the expected value of an <input> or <textarea> element.

Valid attribute values (when used with <input> and <textarea> elements) include:

Value	Description
text	A short hint that describes what is expected as input

poster

[HTMLAttributePoster]

The HTML `poster` attribute indicates the image to be shown while a <video> element is downloading (or until user hits play).

Valid attribute values (when used with <video> elements) include:

Value	Description

URL	URL of file containing image to be shown whilst video is downloading or until user hits play button

preload

[HTMLAttributePreload]

The HTML `preload` attribute indicates if / how the page author thinks <audio> or <video> elements should be loaded when the page loads.

Valid attribute values (when used with <audio> and <video> elements) include:

Value	Description
`auto`	Browser to load entire video when page loads
`metadata`	Browser should only load the video's metadata
`none`	Browser should not load video when page loads

radiogroup

[HTMLAttributeRadiogroup]

The HTML `radiogroup` attribute specifies the commands that are toggled when a <menuitem> element is toggled.

Valid attribute values (when used with <menu> elements) include:

Value	Description
groupname	Name of group of commands toggled when the menu item is toggled. Only applies if type = `radio`

readonly

[HTMLAttributeReadonly]

The HTML `readonly` attribute indicates whether an <input> or <textarea> element is read-only.

Valid attribute values (when used with <input> and <textarea> elements) include:

Value	Description
`readonly`	Indicates that input box is read-only

rel

[HTMLAttributeRel]

The HTML `rel` attribute indicates the relationship between the current document and the document to which it is linked. It applies to <a>, <area> and <link> elements.

Valid attribute values (when used with <a>, <area> and <link> elements) include:

Value	Description
alternate	An alternative representation of the document
author	Link to resource describing author of document
bookmark	URL used for bookmarking (for <a>and <area>)
dns-prefetch	Browser should preemptively do a DNS on the origin of the target (for <link>)
external	Referenced document is not part of same site as original (for <a>)
help	Document providing help
icon	Get icon representing document (for <link>)
license	Copyright information on the document
next	Next document in series
nofollow	An unendorsed document (e.g. a paid link, search spiders may not then follow that link)
noreferrer	Browser should not send an HTTP referrer header if user follows hyperlink
noopener	Browser context created by following hyperlink should not have an opener browser context (for <a>)
pingback	Address of pingback server handling pingbacks relating to origin of document (for <link>)
preconnect	Browser should pre-emptively connect to target (for <link>)
prefetch	Browser should pre-emptively fetch and cache target (for <area>, <link>)
preload	Browser should pre-emptively fetch and render target (for <link>)
prerender	Same as preload for some browsers (for <link>)
prev	Previous document in series
search	A search tool covering the document
stylesheet	Import a CSS stylesheet (for <link>)
tag	A keyword relevant to the current document (for <a>and <area>)

required

[HTMLAttributeRequired]

The HTML required attribute indicates whether an <input>, <select> or <textarea> element needs to be filled in before a form is submitted.

Valid attribute values (when used with <input>, <select> and <textarea> elements) include:

Value	Description
required	Indicates that input box or selection choice must be filled out before the form can be submitted

reversed

[HTMLAttributeReversed]

The HTML reversed attribute indicates whether the list order of an <ol> element should be in descending order (e.g. 3, 2, 1) rather than ascending order (1, 2, 3).

Valid attribute values (when used with <ol> elements) include:

Value	Description

`reversed`	List order is descending

rows

[HTMLAttributeRows]

The HTML `rows` attribute indicates the visible number of lines in a <textarea> element.

Valid attribute values (when used with <textarea> elements) include:

Value	Description
integer	Visible number of rows (lines) in text area

rowspan

[HTMLAttributeRowspan]

The HTML `rowspan` attribute indicates the number of rows a table cell should span. It applies to <td> and <th> elements.

Valid attribute values (when used with <td> and <th> elements) include:

Value	Description
integer	Number of rows a cell should span

sandbox

[HTMLAttributeSandbox]

The HTML `sandbox` attribute indicates extra restrictions applied to the content of an <iframe> element. The sorts of additional restrictions that can be imposed include:

- Deeming the content to come from a unique origin
- Blocking form submission, script execution or execution of APIs
- Preventing some sorts of links
- Preventing content from using plug-ins (e.g. from <embed> or <object> elements)
- Blocking some automatically triggered features (such as automatically playing a <video> element)

Valid attribute values (when used with <iframe> elements) include either `sandbox` (which results in all restrictions being applied) or a space delimited list of values that exclude specific restrictions. These values are:

Value	Description
`allow-forms`	Form submission enabled
`allow-pointer-lock`	APIs allowed
`allow-popups`	Popups allowed
`allow-same-origin`	<iframe> content allowed to be treated as being from same origin as main document

`allow-scripts`	Scripts allowed
`allow-top-navigation`	<iframe> content allowed to navigate to its top-level browsing context
`sandbox` (i.e. *no value*)	All restrictions applied

scope

[HTMLAttributeScope]

The HTML `scope` attribute indicates whether a table header cell (i.e. a <th> element) is a header for a column, a row or for groups of either columns or rows. The `scope` attribute is no longer supported in HTML 5.

Valid attribute values (when used with <td> elements) include:

Value	Description
`col`	Cell is header for a column
`colgroup`	Cell is header for a group of columns
`row`	Cell is header for a row
`rowgroup`	Cell is header for a group of rows

scoped

[HTMLAttributeScoped]

The HTML `scoped` attribute indicates that styles in a <style> element only apply to the element's parent and that element's child elements. The aim is to allow style declarations that apply only to individual parts of a HTML document, but currently it does not always work with major browsers.

Valid attribute values (when used with <style> elements) include:

Value	Description
`scoped`	Styles only apply to element's parent element and that element's child elements

selected

[HTMLAttributeSelected]

The HTML `selected` attribute indicates that an <option> element should be pre-selected when the page loads.

Valid attribute values (when used with <option> elements) include:

Value	Description
`selected`	Option should be pre-selected when page loads

shape

[HTMLAttributeShape]

The HTML `shape` attribute indicates shape of an <area> element. It, together with the coords attribute specify the size, shape and position of the area.

Valid attribute values (applied to <area> elements) include:

Value	Description
`default`	Indicates the entire region
`circle`	Indicates a circular region
`poly`	Indicates a polygonal region
`rect`	Indicates a rectangular region

size

[HTMLAttributeSize]

The HTML `size` attribute indicates the width in characters for <input> elements or number of visible options for <select> elements.

Valid attribute values (when used with <input> elements) include:

Value	Description
integer	Number of characters that identify the width of the element

Valid attribute values (when used with <select> elements) include:

Value	Description
integer	Number of visible options in drop-down menu

sizes

[HTMLAttributeSizes]

The HTML `sizes` attribute specifies the size of a linked resource. It applies to <img>, <link> and <source> elements.

Valid attribute values (when used with <link> elements) include:

Value	Description
heightxwidth	Specifies one or more sizes for linked icon, in the form e.g. `sizes="16x16"` or `sizes="16x16 32x32"`. Is only relevant for `rel=icon`
`any`	Icon is scalable

Note: most browsers do not currently seem to support the `sizes` attribute (at the time of writing it was an experimental attribute for <link> elements). For <source> elements it is only relevant when the <source> element is a direct child of a picture element.

span

[HTMLAttributeSpan]

The HTML `span` attribute specifies the number of columns that a <col> or <colgroup> element spans.

Valid attribute values (for <col>, <colgroup>) include:

Value	Description
integer	Number of columns the element should span

spellcheck

[HTMLAttributeSpellcheck]

The HTML `spellcheck` attribute indicates whether an element is to have its spelling and grammar checked.

Valid attribute values include:

Value	Description
`false`	Element is not to be spellchecked
`true`	Element is to be spellchecked and grammar-checked

src

[HTMLAttributeSrc]

The HTML `src` attribute indicates the URL of a resource. It applies to <audio>, <embed>, <iframe>, <img>, <input>, <script>, <source>, <track> and <video> elements.

Valid attribute values (when used with <audio>, <embed>, <iframe>, <img>, <script>, <source>, <track> and <video> elements) include:

Value	Description
URL	URL of source

Valid attribute values (when used with <input> elements) include:

Value	Description
URL	URL of image to use as a submit button (only for type = `image`)

srcdoc

[HTMLAttributeSrcdoc]

The HTML `srcdoc` attribute indicates the HTML content of the page to be shown in an <iframe> element.

If a browser supports this attribute then it will override the content specified by the src attribute (if present). If it does not support this attribute then it will show the file specified by the src attribute (if present).

The `srcdoc` attribute is usually used in conjunction with the sandbox attribute and the seamless attribute (the seamless attribute is not currently supported by major browsers so is not covered further here).

Valid attribute values (when used with <iframe> elements) include:

Value	Description
HTML_content	HTML content of the page to show in the element

srclang

[HTMLAttributeSrclang]

The HTML `srclang` attribute indicates the language of text data of a <track> element if its `kind` = `subtitles`.

Valid attribute values (when used with <input> elements) include:

Value	Description
language-code	Language of track text data (only for kind = `subtitles`)

srcset

[HTMLAttributeSrcset]

The HTML `srcset` attribute indicates the URL of image to use in different situations for <img> and <source> elements.

Valid attribute values (when used with <img> and <source> elements) include:

Value	Description
URL	URL of image to use in different situations

start

[HTMLAttributeStart]

The HTML `start` attribute indicates the Start value to use for an ordered list (i.e. an <ol> element).

Valid attribute values (when used with <ol> elements) include:

Value	Description
integer	Starting value of list

step

[HTMLAttributeStep]

The HTML `step` attribute indicates the accepted number intervals for an <input> element. For example, if `step="4"` then the accepted numbers could be -4, 0, 4, 8, ….

Valid attribute values (when used with <input> elements) include:

Value	Description
integer	Number of intervals

style

[HTMLAttributeStyle]

The HTML `style` attribute indicates an 'inline' CSS style for that element.

Valid values include:

Value	Description
CSSstyle	A CSS style definition

tabindex

[HTMLAttributeTabindex]

The HTML `tabindex` attribute indicates the tab order of an element, i.e. the order in which the user is taken between elements when pressing the tab key (1 is first element).

In HTML 5 this attribute can be applied to any element.

Valid attribute values include:

Value	Description
integer	Tab order

target

[HTMLAttributeTarget]

The HTML `target` attribute indicates where / how to open a linked document (or where to submit a form). It applies to <a>, <area>, <base> and <form> elements.

Valid attribute values (for <a>, <area>, <base> and <form>) include:

Value	Description
`_blank`	Opens linked document in a new window or tab
`_parent`	Opens linked document in parent frame
`_self`	(default value). Opens linked document in the same window or tab as was clicked
`_top`	Opens linked document in full body of the window

framename	Opens linked document in named frame (not applicable to <form> elements)

title

[HTMLAttributeTitle]

The HTML `title` attribute specifies extra information about an element. Often, if you move the mouse over an element with its title set then the title typically appears as tooltip text next to the element.

In HTML 5 this attribute can be applied to any element.

Valid attribute values include:

Value	Description
text	Tooltip text relating to element

transitionend

[HTMLAttributeTransitionend]

The HTML `transitionend` attribute specifies the event that is triggered when a CSS transition ends. It applies to any element with a CSS transition.

translate

[HTMLAttributeTranslate]

The HTML `translate` attribute specifies whether the content of an element should be translated.

Valid attribute values include:

Value	Description
`yes`	Element content should be translated
`no`	Element content should not be translated

Note: at the time of writing this attribute was not supported by major browsers.

type

[HTMLAttributeType]

The HTML `type` attribute indicates the type of an element. It applies to <area>, <button>, <embed>, <input>, <keygen>, <link>, <menu>, <menuitem>, <object>, <ol>, <script>, <source> and <style> elements.

Valid attribute values (when used with <area>, <embed>, <link>, <object>, <script> elements) include:

Value	Description
media_type	The internet media type (previously known as MIME type) of the target URL

Valid attribute values (when used with <button> elements) include:

Value	Description
`button`	Is a clickable button
`reset`	Is a submit button (so submits form data)
`submit`	Is a reset button (so resets form data to initial / default values)

Valid attribute values (when used with <input> and <keygen> elements) include:

Value	Description
`button`	A clickable button that (typically) activates some JavaScript when clicked
`checkbox`	Input field allowing selection of one or more options from a limited list of options
`color`	Input field for selecting a colour
`date`	Input field for entering a date
`datetime`	Input field for entering a date and time (including time zone) (N.B. Is not currently supported by most browsers)
`datetime-local`	Input field for entering a date and time (no specific time zone)
`email`	E-mail address input field (automatically validated when submitted)
`file`	Define a file selection (with browse) button (for file uploads)
`hidden`	Hidden field (i.e. not visible to user). Is often used to store a default value or may be used as a variable by JavaScript
`image`	Define image as a submit button
`month`	Input field for entering a month and year (no specific time zone)
`number`	Input field for entering a number. The default value is the value attribute. Minimum, maximum and legal number intervals are defined by the min, max and step attributes
`password`	Password field (characters are masked)
`radio`	Buttons allowing user to select only one of a limited number of choices
`range`	Slider control, i.e. a control whose exact values are not important. Default range is 0 to 100 but restrictions can be placed, e.g. using the min, max and step attributes.
`reset`	Reset button (e.g. for resetting all form values to default values)
`search`	A search field
`submit`	Submit button
`tel`	Input field for entering a telephone number
`text`	Single-line input field for entering text
`time`	Input field for entering a time (no specific time zone)
`url`	Input field for entering a URL
`week`	Input field for entering a week and year (no specific time zone)

Valid attribute values (when used with <menu> elements) include:

Value	Description

`list`	A list menu
`toolbar`	A toolbar menu
`context`	A context menu

Valid attribute values (when used with <menuitem> elements) include:

Value	Description
`checkbox`	Command can be toggled using a checkbox
`command`	A normal command
`Radio`	Command can be toggled using a radio button

Note: at the time of writing the `type` attribute for <menuitem> elements was not supported by major browsers.

Valid attribute values (when used with <ol> elements) include:

Value	Description
`1`	List is of type 1, 2, 3, 4, …
`A`	List is of type A, B, C, D, …
`a`	List is of type a, b, c, d, …
`I`	List is of type I, II, III, IV, …
`i`	List is of type i, ii, iii, iv, …

Valid values (when used with <source> elements) include:

Value	Description
MIME-type	MIME type of resource

Valid attribute values (when used with <style> elements) include:

Value	Description
`text/css`	Media type of the <style> element

usemap

[HTMLAttributeUsemap]

The HTML `usemap` attribute indicates whether an <img> or <object> element should be used as a client-side image-map.

Valid attribute values (when used with <img> or <object> elements) include:

Value	Description
#mapname	Name of client-side image-map, i.e. a hash character (#) plus the name of the <map> element to which the `usemap` relates

value

[HTMLAttributeValue]

The HTML `value` attribute indicates the value of an element. It applies to <button>, <data>, <input>, <li>, <meter>, <option>, <progress> and <param> elements.

Valid attribute values (when used with <button>, <input> elements) include:

Value	Description
text	Initial value for button

Note: for some older browsers, if you use a <button> element inside a <form> element then the browser may submit the text between the `<button>` and `</button>` tags rather than the value of its `value` attribute.

Valid attribute values (when used with <data> elements) include:

Value	Description
machine-readable format	Machine-readable content

Valid attribute values (when used with <li> elements) include:

Value	Description
integer	Value of a list item (following list items will increment from that number). Only applicable to lists

Valid attribute values (when used with <meter> elements) include:

Value	Description
number	Value of gauge

Valid attribute values (when used with <option> elements) include:

Value	Description
text	Value to be sent to server

width

[HTMLAttributeWidth]

The HTML `width` attribute indicates the width of an element. It applies to <canvas>, <embed>, <iframe>, <img>, <input>, <object> and <video> elements.

Valid attribute values (when used with <canvas>, <embed>, <iframe>, <img>, <input>, <object> and <video> elements) include:

Value	Description
number	Width of element or embedded content in pixels, e.g. `width="20"`
percentage	Width as a percentage of surrounding element, e.g. `width="30%"`

Note: for some browsers and for some (but not all) of the elements listed above it appears to be necessary if the width is being set in JavaScript to set it using the CSS width property, e.g. in the form *element*`.style.width = "20px"` rather than using the `width` attribute.

wrap

[HTMLAttributeWrap]

The HTML `wrap` attribute indicates how text in a <textarea> element is to be wrapped when submitted in a form.

Valid attribute values (when used with <textarea> elements) include:

Value	Description
`hard`	Text is wrapped (contains newlines) when submitted. The cols attribute must then be specified
`soft`	(default value). Text is not wrapped when submitted

xmlns

[HTMLAttributeXmlns]

The HTML `xmlns` attribute indicates the XML namespace attribute applicable to the webpage (if it needs to conform to XHTML). It applies to <html> elements.

Valid attribute values (when used with <html> elements) include:

Value	Description
`http://www.w3.org/1999/xhtml`	The default XHTML specification

HTML: types of attribute values

[HTMLTypesOfAttributeValues]

Many HTML attributes accept specific types of input, including the following:

Value	Description
#mapname	Name of client-side image-map, i.e. a hash character (#) plus the name of a <map> element
character	A single keyboard character (e.g. as per the accesskey attribute)
character_set	A character encoding (i.e. way of representing characters that will be recognised by a receiving computer), such as: - `UTF-8`: Unicode - `ISO-8859-1`: character encoding for Latin alphabet
CSSclass	Name of a CSS class. These must begin with a letter A-Z or a-z, which can be followed by letters (A-Z or a-z), digits (0-9), hypens ("-") and underscores ("_"). In HTML all such values are case-insensitive, i.e. class names of "abc" and "ABC" are treated as synonymous.
CSSstyle	A CSS style definition
datalist_id	Id (identifier) of relevant <datalist> element
date	A date
elementID	The id (identifier) defining the associated element
file_extension	A file extension starting with a full stop, e.g. `.png`, `.jpg`, `.pdf`, `.doc`

	(e.g. used for the accept attribute)
filename	File name of a resource
formID	The id defining the associated form
framename	A named frame (i.e. <iframe> element)
groupname	Name of group of commands
header_id	Id of a header cell
heightxwidth	One or more sizes (in pixels), in the form e.g. `sizes="16x16"` or `sizes="16x16 32x32"`.
HTML_content	HTML content
HTTPmethod	Either get or post. These have the following characteristics: - `get`. Use the HTTP 'get' method. This includes the form-data in the URL in name/value pairs. The length of the URL is limited and hence the values transmitted will be public (even if the website is accessed using an https call), but users can bookmark the resulting call - `post`. Use the HTTP 'post' method. This includes the form-data in the HTTP request, i.e. not in the URL, and is not subject to the same size limitations as the 'get' method, but cannot then be bookmarked by users
inputfieldname	Name of an input field
integer	An integer
language-code	Language of text in linked document. The language code is either an ISO 639-1 two letter language code (e.g. "en") or such a code followed by a dash and then a two letter ISO country code (the latter can be used if different countries recognise different versions of the same language, e.g. "en-gb" versus "en-us")
machine-readable format	Machine-readable content
media-query	Media or device type
media_type	A valid media type, see e.g. http://www.iana.org/assignments/media-types/media-types.xhtml (e.g. as per accept attribute)
MIME-type	MIME type of resource
name	Name of element, attribute or (for <meta> elements) metadata item
no value	I.e. element should be left black, however see below regarding attribute minimisation and XHTML.
number	A number (sometimes only an integer is acceptable, e.g. for the cols attribute, sometimes a floating-point value is also acceptable), usually in the form of a string (enclosed in quotes) representing the number
percentage	A percentage, e.g. `30%`
regular-expression	Regular expression (against which e.g. an input is compared)
text	Text
URI	Uniform Resource Identifier, see below.
URL	i.e. Uniform Resource Locator. These can be: - Absolute, pointing to a specific webpage, e.g. http://www.nematrian.com/Introduction.aspx, or - Relative, pointing to a file relative to some base, usually the directory or website within which the page accessing the URL is position, e.g. example.htm
x1, y1, x2, y2	Typically involve coordinates as per the coords attribute
YYYY-MM-	A date (in a specific machine and location independent format)

DDThh:mm:ssTZD	

When a value the attribute can take is shown as the same as its own name then this is a Boolean-style attribute, meaning that in HTML the attribute should either be mentioned (but assigned no value), in which case the attribute applies, or be absent, in which case the attribute does not apply. In XHTML, such *attribute minimisation* is not allowed, and the attribute needs to be defined explicitly, taking as its value its own name, e.g. `<video … autoplay="autoplay">…</video>`.

Event attributes (which usually have the form `on…`) take values which are JavaScript functions.

Uniform Resource Identifiers (URIs):

'URI' stands for 'Uniform Resource Identifier'. The possible set of parts a URI can contain are illustrated by the following:

```
http://username:pword@www.example.org:80/path/file.aspx?a=23&b=has+spaces#anchor
```

A URI encoded string has each instance of certain characters replaced by one, two, three or (rarely) four escape sequences representing the UTF-8 encoding of the character. `encodeURI` escapes all characters except for the following (so it does not encode characters needed to formulate a complete URI as above, or a few additional 'unreserved marks' which do not have a reserved purpose as such and are allowed in a URI 'as is'):

```
A-Z a-z 0-9 ; , / ? : @ & = + $ - _ . ! ~ * ' ( ) #
```

There are four global JavaScript methods that convert strings into URIs and vice versa (six if depreciated methods are included).

`encodeURI()` escapes all characters except for the following (so it does not encode characters needed to formulate a complete URI as above, or a few additional 'unreserved marks' which do not have a reserved purpose as such and are allows in a URI 'as is':

```
A-Z a-z 0-9 ; , / ? : @ & = + $ - _ . ! ~ * ' ( ) #
```

`encodeURIComponent()` also escapes reserved characters, so escapes all characters except:

```
A-Z a-z 0-9 - _ . ! ~ * ' ( ) #
```

`escape()` (now depreciated, use `encodeURI` or `encodeURIComponent` instead) encodes all characters with the exception of `* @ - _ + . /`

`decodeURI()`, `decodeURIComponent()` and `unescape()` are the inverses of `encodeURI()`, `encodeURIComponent()` and `escape()` respectively.

If you want to encode a string but avoid encoding square brackets (these are becoming reserved characters for IPv6) then it is recommended that you use a JavaScript statement like:

```
encode(str).replace(/%5B/g,'[').replace(%5D/g,']')
```

Appendix C: CSS Properties

[CSSProperties]

Set out below are different CSS properties (and rules):

Property	Description	More	Type
align-content	Modifies the behaviour of the flex-wrap property, aligning flex lines	Here	Flexible Container
align-items	Specifies the default alignment for items inside a flexible container	Here	Flexible Container
align-self	Specifies the alignment for selected item in a flexible container	Here	Flexible Container
all	Resets (almost) all properties to their initial or inherited values	Here	All
animation	A shorthand property combining (up to) 8 individual animation properties	Here	Animation
animation-delay	Specifies delay until start of animation	Here	Animation
animation-direction	Indicates direction of animation	Here	Animation
animation-duration	Indicates time an animation takes to play	Here	Animation
animation-fill-mode	Style element takes when the animation is not playing	Here	Animation
animation-iteration-count	Number of times an animation should play	Here	Animation
animation-name	Name of animation used in a @keyframes animation	Here	Animation
animation-play-state	Whether the animation is paused or not	Here	Animation
animation-timing-function	The speed curve of an animation	Here	Animation
backface-visibility	Whether element should remain visible when not facing the screen	Here	Transform
background	A shorthand property combining (up to) 8 background properties	Here	Background
background-attachment	Whether background image is fixed or scrolls with rest of page	Here	Background
background-blend-mode	The blending mode of each background layer	Here	Background
background-clip	Painting area of the background	Here	Background
background-color	Background colour of an element	Here	Background
background-image	One or more background images for element	Here	Background
background-origin	Where the background image for element is positioned	Here	Background
background-position	The (starting) position of a background image	Here	Background

background-repeat	How background image repeated	Here	Background
background-size	Size of background image(s)	Here	Background
border	A shorthand property combining (up to) 3 border properties	Here	Border
border-bottom	A shorthand property combining all main bottom border properties	Here	Border
border-bottom-color	Colour of bottom border of element	Here	Border
border-bottom-left-radius	Shape of the bottom-left corner of a border	Here	Border
border-bottom-right-radius	Shape of the bottom-right corner of a border	Here	Border
border-bottom-style	Style of bottom border of element	Here	Border
border-bottom-width	Width of bottom border of element	Here	Border
border-collapse	Whether table borders are collapsed	Here	Table
border-color	The colour of an element's four borders	Here	Border
border-image	A shorthand property combining (up to) 5 border-image properties	Here	Border
border-image-outset	Amount by which a border image area extends beyond the border box	Here	Border
border-image-repeat	How border image should be repeated, rounded or stretched	Here	Border
border-image-slice	How any border image should be sliced	Here	Border
border-image-source	Path of image to be used as a border	Here	Border
border-image-width	Width of border image	Here	Border
border-left	A shorthand property combining all main left border properties	Here	Border
border-left-color	Colour of left border of element	Here	Border
border-left-style	Style of left border of element	Here	Border
border-left-width	Width of left border of element	Here	Border
border-radius	Adds rounded corners to element	Here	Border
border-right	A shorthand property combining all main right border properties	Here	Border
border-right-color	Colour of right border of element	Here	Border
border-right-style	Style of right border of element	Here	Border
border-right-width	Width of right border of element	Here	Border
border-spacing	Distance between borders of adjacent cells (if border-collapse property is separate)	Here	Table
border-style	Style of an element's four borders	Here	Border

`border-top`	A shorthand property combining all main top border properties	Here	Border
`border-top-color`	Colour of top border of element	Here	Border
`border-top-left-radius`	Shape of the top -left corner of a border	Here	Border
`border-top-right-radius`	Shape of the top -right corner of a border	Here	Border
`border-top-style`	Style of top border of element	Here	Border
`border-top-width`	Width of top border of element	Here	Border
`border-width`	Width of an element's four borders	Here	Border
`bottom`	Bottom edge of element relative to the corresponding edge of nearest positioned ancestor	Here	Box
`box-shadow`	Applies one or more shadows to element	Here	Border
`box-sizing`	What box sizing (height, width) should be applied to	Here	User Interface
`caption-side`	Where a table caption should be placed	Here	Table
`clear`	Which sides a floating element is not allowed to float	Here	Box
`clip`	What to do with an image that is larger than its containing element	Here	Box
`color`	Colour of text within an element	Here	Colour
`column-count`	Number of columns element should be divided into	Here	Column Layout
`column-fill`	How to fill columns	Here	Column Layout
`column-gap`	What gap to place between columns	Here	Column Layout
`column-rule`	A shorthand property combining (up to) 3 column-rule properties	Here	Column Layout
`column-rule-color`	Colour of any rule between columns	Here	Column Layout
`column-rule-style`	Style of any rule between columns	Here	Column Layout
`column-rule-width`	Width of any rule between columns	Here	Column Layout
`column-span`	How many columns an element should span across	Here	Column Layout
`column-width`	Suggested optimal width for columns	Here	Column Layout
`columns`	A shorthand property for setting certain column properties	Here	Column Layout
`content`	Pseudo-property used with the `:before` and `:after` pseudo-elements to insert generated content	Here	User Interface
`counter-increment`	Pseudo-property that increments one or more CSS counter values	Here	Counters
`counter-reset`	Pseudo-property that creates or resets one or more CSS counter values	Here	Counters
`cursor`	Type of cursor to be displayed	Here	User Interface
`direction`	Text or writing direction	Here	Text

`display`	Type of box to be used for element	Here	Box
`empty-cells`	Whether to display borders and background for empty table cells	Here	Table
`filter`	Applies visual effects like grayscale, blur and saturation	Here	Images
`flex`	A shorthand property for setting certain flex properties	Here	Flexible Container
`flex-basis`	Initial length of a flexible element	Here	Flexible Container
`flex-direction`	Direction of a flexible element	Here	Flexible Container
`flex-flow`	A shorthand property for setting certain flex properties	Here	Flexible Container
`flex-grow`	How much an element will grow relative to other flexible items in a container	Here	Flexible Container
`flex-shrink`	How much an element will shrink relative other flexible items in a container	Here	Flexible Container
`flex-wrap`	Whether flexible items should wrap	Here	Flexible Container
`float`	Whether element should float	Here	Box
`font`	A shorthand property for setting font properties	Here	Font
`@font-face`	A rule allowing designers to apply their own font	Here	Font, Rules
`font-family`	Font to be used	Here	Font
`font-size`	Size of font to be used	Here	Font
`font-size-adjust`	Refined sizing of font to be used	Here	Font
`font-stretch`	Makes text in an element narrower or more stretched out than usual	Here	Font
`font-style`	Font style to be used	Here	Font
`font-variant`	Whether text should be in small-caps font	Here	Font
`font-weight`	How thick or thin (i.e. bold or not) characters should be	Here	Font
`hanging-punctuation`	Whether a punctuation mark can be placed outside box at start or end of full line of text	Here	Text
`height`	Height of an element (excluding padding, borders and margins)	Here	Box
`justify-content`	How to align a flexible container's items when the items do not use all available space along the horizontal axis	Here	Flexible Container
`@keyframes`	A rule allowing designers to specify animations	Here	Animation, Rules
`left`	Left edge of element relative to the corresponding edge of nearest positioned ancestor	Here	Box
`letter-spacing`	Amount of space between consecutive text characters	Here	Text
`line-height`	Height of lines of text	Here	Text
`list-style`	A shorthand property combining up to 3	Here	List

	list properties		
`list-style-image`	image to use as the list-item marker	Here	List
`list-style-position`	Whether a list marker is inside or outside the relevant content container	Here	List
`list-style-type`	Type of list-item marker	Here	List
`margin`	A shorthand property combining all four margin properties	Here	Margin
`margin-bottom`	Width of bottom margin of element	Here	Margin
`margin-left`	Width of left margin of element	Here	Margin
`margin-right`	Width of right margin of element	Here	Margin
`margin-top`	Width of top margin of element	Here	Margin
`max-height`	Maximum height element can become	Here	Box
`max-width`	Maximum width element can become	Here	Box
`@media`	A rule allowing designers to apply different styles for different devices and/or media types	Here	Rule
`min-height`	Minimum height element can become	Here	Box
`min-width`	Minimum width element can become	Here	Box
`nav-down`	Where to navigate with down arrow key	Here	User Interface
`nav-index`	The sequential navigation order (i.e. the 'tabbing order') for an element	Here	User Interface
`nav-left`	Where to navigate with left arrow key	Here	User Interface
`nav-right`	Where to navigate with right arrow key	Here	User Interface
`nav-up`	Where to navigate with up arrow key	Here	User Interface
`opacity`	Degree of opacity (transparency)	Here	Colour
`order`	Order of a flexible item relative to other flexible items inside the same container	Here	Flexible Container
`orphans`	Minimum number of lines of a paragraph in a paged media that can be left on an old page	Here	Paged media
`outline`	A shorthand property combining (up to) 3 outline properties	Here	User Interface
`outline-color`	Colour of outline	Here	User Interface
`outline-offset`	amount of space between an element's outline and edge or border of element	Here	User Interface
`outline-style`	Style to outline	Here	User Interface
`outline-width`	Width to outline	Here	User Interface
`overflow`	What happens when content overflows an element's box	Here	Box
`overflow-x`	What to with left/right edges of content overflowing an element's box	Here	Box
`overflow-y`	What to with top/bottom edges of content overflowing an element's box	Here	Box
`padding`	A shorthand property combining the 4 padding sub-properties	Here	Padding
`padding-bottom`	Width of the bottom padding	Here	Padding
`padding-left`	Width of the left padding	Here	Padding
`padding-right`	Width of the right padding	Here	Padding

padding-top	Width of the top padding	Here	Padding
page-break-after	Whether a page break should occur after element	Here	Page Media
page-break-before	Whether a page break should occur before element	Here	Page Media
page-break-inside	Whether a page break allowed inside element	Here	Page Media
perspective	How far 3D element is notionally placed behind the screen	Here	Transform
perspective-origin	Where 3D element is notionally placed	Here	Transform
position	How element should be positioned	Here	Box
quotes	How quotation marks should be rendered	Here	Text
resize	Whether element can be resized by user	Here	User Interface
right	Right edge of element relative to the corresponding edge of nearest positioned ancestor	Here	Box
tab-size	Size of space used for tab character	Here	
table-layout	Algorithm used to define table layout	Here	Table
text-align	How text in element should be aligned	Here	Text
text-align-last	How last line of text in element should be aligned	Here	Text
text-decoration	The 'decoration' (e.g. underlining) added to text	Here	Text Decoration
text-decoration-color	Colour of text decoration added to text	Here	Text Decoration
text-decoration-line	Line type of text decoration added to text	Here	Text Decoration
text-decoration-style	Line style of text decoration added to text	Here	Text Decoration
text-indent	Indentation applied to first line of text	Here	Text
text-justify	Type of justification applied to first line of text (if applicable)	Here	Text
text-overflow	How text that has overflowed is to be rendered by browser	Here	Text, User Interface
text-shadow	What shadow should be added to text	Here	Text Decoration
text-transform	Capitalisation to use for text	Here	Text
top	Top edge of element relative to the corresponding edge of nearest positioned ancestor	Here	Box
transform	Applies 2D or 3D transformation	Here	Transform
transform-origin	Origin used by the transform property	Here	Transform
transform-style	How nested elements are to be rendered for 3D purposes when using the transform property	Here	Transform
transition	A shorthand property combining the 4 transition sub-properties	Here	Transitions

transition-delay	When transition should start	Here	Transitions
transition-duration	How long transition will take to complete	Here	Transitions
transition-property	Properties that change as part of a transition effect	Here	Transitions
transition-timing-function	Speed curve used for a transition effect	Here	Transitions
unicode-bidi	Whether text direction should be overridden to support multiple languages	Here	Text
user-select	Whether text of element can be selected	Here	Text
vertical-align	Vertical alignment of element	Here	Box
visibility	Whether element is visible	Here	Box
white-space	How white-space inside element is handled	Here	Text
widows	Minimum number of lines of a paragraph in a paged media that can fall to a new page	Here	Paged media
width	Width of an element (excluding padding, borders and margins)	Here	Basic
word-break	Way in which words can be broken at line ends for non-CJK scripts	Here	Text
word-spacing	Amount of whitespace between words	Here	Text
word-wrap	Whether long words can be broken at line ends and wrap onto the next line	Here	Text
z-index	Stack order of an element, i.e. whether it is "in front of" other elements, and hence is visible if several would otherwise appear in the same place	Here	Box

At the time of writing there were also some other CSS properties and rules being developed by some organisations including:

- box-decoration-break
- break-after
- break-before
- break-inside
- @font-feature-values
- font-feature-settings
- font-kerning
- font-language-override
- font synthesis
- font-variant-alternatives
- font-variant-caps
- font-variant-east-asian
- font-variant-ligatures
- font-variant-numeric
- font-variant-position
- hyphens
- icon (is not currently supported by many major browsers)
- image-orientation

- image-rendering
- image-resolution
- ime-mode
- line-break
- mark
- mark-after
- mark-before
- marks
- marquee-direction
- marquee-play-count
- marquee-speed
- marquee-style
- mask
- mask-type
- object-fit
- object-position
- orphans
- overflow-wrap
- phonemes
- rest
- rest-after
- rest-before
- text-combine-upright
- text-orientation
- text-underline-position
- voice-balance
- voice-duration
- voice-pitch
- voice-pitch-range
- voice-rate
- voice-stress
- voice-volume
- widows
- writing-mode

Individual CSS Properties:

align-content

[CSSPropertyAlignContent]

The CSS (CSS3) `align-content` property modifies the behaviour of the flex-wrap property, aligning flex lines. N.B. If you want to align items on the main *x*-axis then use the justify-content property.

Valid property values (other than inherit and initial) are:

Value	Description
`center`	Lines packed towards centre of flex container
`flex-end`	Lines packed towards end of flex container
`flex-start`	Lines packed towards start of flex container
`space-between`	Lines distributed evenly in flex container
`space-around`	Lines distributed evenly in flex container but with half-size spaces at either end
`stretch`	(default value). Lines stretch to take up remaining space

Default Value: `stretch`
JavaScript syntax: e.g. *object*`.style.alignContent = "center";`
Inherited: No
Animatable**:** No

align-items

[CSSPropertyAlignItems]

The CSS (CSS3) `align-items` property specifies the default alignment for items inside a flexible container. Use the align-self property to override the property for any individual item.

Valid property values (other than inherit and initial) are:

Value	Description
`baseline`	Items positioned at baseline of container
`center`	Items positioned at centre of container
`flex-end`	Items positioned at end of container
`flex-start`	Items positioned at start of container
`stretch`	(default value). Items stretched to fit container

Default Value: `stretch`
JavaScript syntax: e.g. *object*`.style.alignItems = "center";`
Inherited: No
Animatable**:** No

align-self

[CSSPropertyAlignSelf]

The CSS (CSS3) `align-self` property specifies the alignment for the selected item within a flexible container. Use the align-items property to set the default that otherwise applies to the items in the flexible container.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Element inherits its parent container's align-items property, or "stretch" if it has no parent container
`baseline`	Items positioned at baseline of container
`center`	Items positioned at centre of container
`flex-end`	Items positioned at end of container
`flex-start`	Items positioned at start of container
`stretch`	Items stretched to fit container

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.alignSelf = "center";`
Inherited: No
Animatable**:** No

all

[CSSPropertyAll]

The CSS (CSS3) `all` property resets all properties, apart from unicode-bidi and direction, to their initial or inherited values.

Valid property values (other than inherit and initial) are:

Value	Description
`unset`	Changes all properties applied to element or its parent to their parent value if they are inheritable or to their initial value if not

Default Value: N/A
JavaScript syntax: e.g. *object*`.style.all = "initial";`
Inherited: No
Animatable**:** No

animation

[CSSPropertyAnimation]

The CSS (CSS3) `animation` property is a shorthand property combining (up to) 8 of the animation properties. N.B. Always specify a non-zero animation-duration property as otherwise the duration will be set to zero and the animation will in effect not play.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- animation-name

- animation-duration
- animation-timing-function
- animation-delay
- animation-iteration-count
- animation-direction
- animation-fill-mode
- animation-play-state

Default Value: *See individual properties*
JavaScript syntax: e.g. *object*`.style.animation = "mymovie 5s infinite";`
Inherited: No
Animatable: No

animation-delay

[CSSPropertyAnimationDelay]

The CSS (CSS3) `animation-delay` property specifies the delay until the start of an animation.

Valid property values (other than inherit and initial) are:

Value	Description
time	Length of time (in CSS time units) to start of animation

Default Value: `0s`
JavaScript syntax: e.g. *object*`.style.animationDelay="2s"`
Inherited: No
Animatable: No

animation-direction

[CSSPropertyAnimationDirection]

The CSS (CSS3) `animation-direction` property defines whether an animation should play in forward, reverse or alternating directions.

Valid property values (other than inherit and initial) are:

Value	Description
`alternate`	Will play in forward direction every odd time and reverse direction every even time
`alternate-reverse`	Will play in reverse direction every odd time and forward direction every even time
`normal`	(default value). Will play in forward direction
`reverse`	Will play in reverse direction

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.animationDirection = "reverse"`
Inherited: No
Animatable: No

animation-duration

[CSSPropertyAnimationDuration]

The CSS (CSS3) `animation-duration` property indicates the length of time an animation takes to play.

Valid property values (other than inherit and initial) are:

Value	Description
time	A CSS time

Default Value: `0s`
JavaScript syntax: e.g. *object*`.style.animationDuration = "5s";`
Inherited: No
Animatable**:** No

animation-fill-mode

[CSSPropertyAnimationFillMode]

The CSS (CSS3) `animation-fill-mode` property specifies the style (for an element involved in an animation) that the element takes when the animation is not playing (i.e. when it is finished, or when it has a delay) defines how many seconds an animation should take to complete one cycle, defined in seconds (s) or milliseconds (ms).

Valid property values (other than inherit and initial) are:

Value	Description
`backwards`	Before start (i.e. during any delay) will apply property values applicable at the start of the time the animation is running
`both`	Will follow both the forwards and backwards rules
`forwards`	After animation is ended will apply property values applicable at the end of the time the animation is running
`none`	(default value). Will not apply any styles before or after animation is executing

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.animationFillMode = "forwards";`
Inherited: No
Animatable**:** No

animation-iteration-count

[CSSPropertyAnimationIterationCount]

The CSS (CSS3) `animation-iteration-count` property specifies the number of times an animation should play.

Valid property values (other than inherit and initial) are:

Value	Description
number	(integer). Number of times an animation should be played
`infinite`	Indicates animation should be played an infinite number of times

Default Value: `1`
JavaScript syntax: e.g. *object*`.style.animationIterationCount = "infinite";`
Inherited: No
Animatable**:** No

animation-name

[CSSPropertyAnimationName]

The CSS (CSS3) `animation-name` property specifies the name of an animation used in a @keyframes animation.

Valid property values (other than inherit and initial) are:

Value	Description
keyframename	Specifies name of the keyframe you want to bind to the selector
`none`	(default value). Indicates no animation

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.animationName = "myanimation";`
Inherited: No
Animatable**:** No

The keyframes format is e.g.:

```
@keyframes myanimation {
    from { starting style }
    to { ending style }
}
```

animation-play-state

[CSSPropertyAnimationPlayState]

The CSS (CSS3) `animation-play-state` property specifies whether the animation is paused or not. One of its uses is to pause an animation in the middle of a cycle using JavaScript.

Valid property values (other than inherit and initial) are:

Value	Description
`paused`	Specifies animation is paused
`running`	(default value). Specifies animation is running

Default Value: `running`
JavaScript syntax: e.g. *object*`.style.animationPlayState = "paused";`

Inherited: No
Animatable: No

animation-timing-function

[CSSPropertyAnimationTimingFunction]

The CSS (CSS3) `animation-timing-function` property specifies the speed curve of an animation. The speed curve is defined by a cubic Bezier curve.

Valid property values (other than inherit and initial) are:

Value	Description
`cubic-bezier`(*x1*,*x2*,*x3*,*x4*)	As defined by cubic Bezier function with parameters *x1*, *x2*, *x3*, *x4* (possible values of each are numerical values from 0 to 1)
`ease`	(default value). Slow start, then fast, before ending slowly
`ease-in`	Slow start
`ease-out`	Slow end
`ease-in-out`	Slow start and end
`linear`	Animation has same speed throughout its life
`step-start`	Equivalent to steps(1, start)
`step-end`	Equivalent to steps(1, end)
`steps`(*int*, *start*|*end*)	A stepping function with two parameters. The first parameter specifies the number of intervals in the function (and must be a positive integer, i.e. greater than zero). The second (optional) parameter specifies when the change of values occurs and is either "start" or "end" (if omitted is given the value "end")

Default Value: `ease`
JavaScript syntax: e.g. *object*`.style.animationTimingFunction="linear"`
Inherited: No
Animatable: No

backface-visibility

[CSSPropertyBackfaceVisibility]

The CSS (CSS3) `backface-visibility` property indicates whether an element should remain visible when it is not facing the screen (i.e. what happens when an element is rotated to face away from the viewer).

Valid property values (other than inherit and initial) are:

Value	Description
`hidden`	Backside is not visible
`visible`	(default value). Backside is visible

Default Value: `visible`
JavaScript syntax: e.g. *object*`.style.backfaceVisibility="hidden"`
Inherited: No

Animatable: No

background

[CSSPropertyBackground]

The CSS (CSS1 and CSS3) `background` property is a shorthand property combining (up to) 8 of the background properties.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- background-color
- background-image
- background-position
- background-size
- background-repeat
- background-origin
- background-clip
- background-attachment

Default Value: *See individual properties*
JavaScript syntax: e.g. *object*`.style.background="red url(mypicture.gif) top left no-repeat"`
Inherited: No
Animatable: See *individual properties*

If one of the properties in the shorthand declaration is the background-size property then you need to use a "/" to separate it from the background-position property (the background-size property was added in CSS3), e.g.:

```
div {background: url(mypicture.gif) 10px 20px/40px 40px;}
```

More than one background-image can be specified. However, if you are using multiple background-image sources and also want a background-color then you need to put the background-color parameter last in the list.

background-attachment

[CSSPropertyBackgroundAttachment]

The CSS (CSS1) `background-attachment` property indicates whether a background image is fixed or scrolls with the rest of the page.

Valid property values (other than inherit and initial) are:

Value	Description
`fixed`	Background is fixed in relation to viewport
`local`	Background scrolls along with the element's contents
`scroll`	(default value). Background scrolls along with the element

Default Value: `scroll`
JavaScript syntax: e.g. *object*`.style.backgroundAttachment="fixed"`
Inherited: No
Animatable: No

background-blend-mode

[CSSPropertyBackgroundBlendMode]

The CSS (CSS3) `background-blend-mode` property defines the blending mode of each background layer (i.e. colour and/or image).

Valid property values are:

Value	Description
`color`	Blending mode set to color
`color-dodge`	Blending mode set to color-dodge
`darken`	Blending mode set to darken
`lighten`	Blending mode set to lighten
`luminosity`	Blending mode set to luminosity
`multiply`	Blending mode set to multiply
`normal`	(default value). Blending mode set to normal
`overlay`	Blending mode set to overlay
`saturation`	Blending mode set to saturation
`screen`	Blending mode set to screen

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.backgroundBlendMode="screen"`
Inherited: No
Animatable: No

background-clip

[CSSPropertyBackgroundClip]

The CSS (CSS3) `background-clip` property specifies the painting area of the background.

Valid property values (other than inherit and initial) are:

Value	Description
`border-box`	(default value). Background is clipped to the border box
`content-box`	Background is clipped to the content box
`padding-box`	Background is clipped to the padding box

Default Value: `border-box`
JavaScript syntax: e.g. *object*`.style.backgroundClip="content-box"`
Inherited: No
Animatable: No

background-color

[CSSPropertyBackgroundColor]

The CSS (CSS1) `background-color` property sets the background colour of an element. The background of an element includes its padding and border but *not* its margin. Usually, a background colour and text colour should be chosen in tandem to make the text easy to read.

Valid property values (other than inherit and initial) are:

Value	Description
color	Specified CSS colour
`transparent`	(default value). Transparent

Default Value: `transparent`
JavaScript syntax: e.g. *object*`.style.backgroundColor="red"`
Inherited: No
Animatable**:** Yes

background-image

[CSSPropertyBackgroundImage]

The CSS (CSS1) `background-image` property sets one or more background images for an element. The background of an element includes its padding and border but *not* its margin. Usually, you should set a background-color to be used if the image is unavailable and the background image and text colour should be chosen in tandem to make the text easy to read.

By default, a background image is placed at the top-left corner of an element and is repeated both vertically and horizontally. These features can be overridden using the background-position and background-repeat properties.

Valid property values (other than inherit and initial) are:

Value	Description
`url(imagefile)`	The URL of the image. To specify more than one image, separate the URLs with a comma
`none`	(default value). No background image supplied

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.backgroundImage="url(mypicture.gif)"`
Inherited: No
Animatable**:** No

background-origin

[CSSPropertyBackgroundOrigin]

The CSS (CSS3) `background-origin` property specifies where the background image for an element is positioned. If the background-attachment property is set to "fixed" then the background-origin property has no effect.

Valid property values (other than inherit and initial) are:

Value	Description
`border-box`	Background image starts from the upper-left corner of the border
`content-box`	Background image starts from the upper-left corner of the content
`padding-box`	(default value). Background image starts from the upper-left corner of the padding edge

Default Value: `padding-box`
JavaScript syntax: e.g. *object*`.style.backgroundOrigin="content-box"`
Inherited: No
Animatable**:** No

background-position

[CSSPropertyBackgroundPosition]

The CSS (CSS1) `background-position` property sets the (starting) position of a background image. By default, a background-image is placed at the top-left corner of an element and is repeated both vertically and horizontally.

Valid property values (other than inherit and initial) are:

Value	Description
value	See below
x% y%	Horizontal position and vertical position as fraction of size of element. Top left is `0% 0%`, bottom right is `100% 100%`. If you only specify one value then the other will be `50%`.
xpos ypos	(e.g. 0px 0px). Horizontal position and vertical position in any valid CSS length. You can mix % and positions

Default Value: `0% 0%`
JavaScript syntax: e.g. *object*`.style.backgroundPosition="10% 20%"`
Inherited: No
Animatable**:** Yes

Other acceptable values include combinations of `left`, `center`, `right` (for *xpos*) and `top`, `center`, `bottom` (for *ypos*). If you only specify one keyword then the other value will be `center`.

background-repeat

[CSSPropertyBackgroundRepeat]

The CSS (CSS1) `background-repeat` property sets whether/how a background image will be repeated. By default, a background-image is placed at the top-left corner of an element and is repeated both vertically and horizontally.

Valid property values (other than inherit and initial) are:

Value	Description
`no-repeat`	Not repeated
`repeat`	(default value). Repeated both vertically and horizontally
`repeat-x`	Repeated only horizontally
`repeat-y`	Repeated only vertically

Default Value: `repeat`
JavaScript syntax: e.g. *object*`.style.backgroundRepeat="repeat-x"`
Inherited: No
Animatable**:** No

background-size

[CSSPropertyBackgroundSize]

The CSS (CSS3) `background-size` property specifies the size of the background image(s).

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Background-image contains its width and height
`contain`	Scales the image to be as large as possible such that both width and height fit inside the content area
`cover`	Scales the image to be as large as possible to cover the background area completely
length	Sets width and height in that order using any valid CSS length. If only one is given the second is set to `auto`
percentage	Sets width and height in that order as percentages of the parent element. If only one is given the second is set to `auto`

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.backgroundSize="60px 80px"`
Inherited: No
Animatable**:** Yes

`

border

[CSSPropertyBorder]

The CSS (CSS1) `border` property is a shorthand property combining (up to) 3 of the border properties.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- border-width
- border-style

- border-color

Default Value: *See individual properties*
JavaScript syntax: e.g. *object*`.style.border="2px solid red"`
Inherited: No
Animatable**:** See *individual properties*

border-bottom

[CSSPropertyBorderBottom]

The CSS (CSS1) `border-bottom` property is a shorthand property combining all the main bottom border properties.

Valid property values (other than inherit or initial) are defined by the elements of the shorthand and are:

- border-bottom-width
- border-bottom-style
- border-bottom-color

Missing properties are given their default values.

Default Value: `medium none` *color*
JavaScript syntax: e.g. *object*`.style.borderBottom="2px solid red"`
Inherited: No
Animatable**:** Yes

In the default value, *color* is the CSS colour of the element.

border-bottom-color

[CSSPropertyBorderBottomColor]

The CSS (CSS1) `border-bottom-color` property sets the colour of the bottom border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
color	Specified CSS colour
`transparent`	Transparent

Default Value: The current CSS colour of the element
JavaScript syntax: e.g. *object*.style`.borderBottomColor="red"`
Inherited: No
Animatable**:** Yes

border-bottom-left-radius

[CSSPropertyBorderBottomLeftRadius]

The CSS (CSS3) `border-bottom-left-radius` property determines the shape of the bottom-left corner of a border. It allows you to add rounded borders to elements.

The two length or percentage values define the radii of a quarter ellipse defining the shape of the corner. The first is the horizontal radius, the second the vertical radius. If either is omitted then it is copied from the other (so the corner is a quarter-circle). If either is zero then the corner becomes square. Percentages for the horizontal radius refer to the width of the border box, those for the vertical radius refer to the height of the border box.

Valid property values (other than inherit and initial) are:

Value	Description
length1, length2	Specified CSS lengths
%, %	See above

Default Value: 0
JavaScript syntax: e.g. *object*`.style.borderBottomLeftRadius="5px 8px"`
Inherited: No
Animatable**:** Yes

border-bottom-right-radius

[CSSPropertyBorderBottomRightRadius]

The CSS (CSS3) `border-bottom-right-radius` property determines the shape of the bottom-right corner of a border. It allows you to add rounded borders to elements.

The two length or percentage values define the radii of a quarter ellipse defining the shape of the corner. The first is the horizontal radius, the second the vertical radius. If either is omitted then it is copied from the other (so the corner is a quarter-circle). If either is zero then the corner becomes square. Percentages for the horizontal radius refer to the width of the border box, those for the vertical radius refer to the height of the border box.

Valid property values (other than inherit and initial) are:

Value	Description
length1, length2	Specified CSS lengths
%, %	See above

Default Value: 0
JavaScript syntax: e.g. *object*`.style.borderBottomRightRadius="5px 8px"`
Inherited: No
Animatable**:** Yes

border-bottom-style

[CSSPropertyBorderBottomStyle]

The CSS (CSS1) `border-bottom-style` property sets the border style of the bottom border of an element.

Valid property values (other than inherit and initial) are shown here.

Default Value:	`none`
JavaScript syntax:	e.g. *object*`.style.borderBottomStyle="groove"`
Inherited:	No
Animatable:	No

border-bottom-width

[CSSPropertyBorderBottomWidth]

The CSS (CSS1) `border-bottom-width` property sets the width of the bottom border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value:	`medium`
JavaScript syntax:	e.g. *object*`.style.borderBottomWidth="4px"`
Inherited:	No
Animatable:	Yes

border-collapse

[CSSPropertyBorderCollapse]

The CSS (CSS2) `border-collapse` property indicates whether table borders are collapsed into a single border or detached as in standard HTML. Note if a !DOCTYPE is not specified then the border-collapse property can produce unexpected results.

Valid property values (other than inherit and initial) are:

Value	Description
`collapse`	Borders are collapsed into a single border where possible (ignoring border-spacing and empty-cells properties)
`separate`	(default value). Borders are detached, and border-spacing and empty-cells properties have some meaning

Default Value:	`separate`
JavaScript syntax:	e.g. *object*`.style.borderCollapse="collapse"`

Inherited: Yes
Animatable: No

border-color

[CSSPropertyBorderColor]

The CSS (CSS1) `border-color` property sets the colour of an element's four borders. The individual border colours can be set separately using border-bottom-color, border-left-color, border-right-color and border-top-color. As with some other aggregate edge properties, up to four parameter values can be supplied (and if more than one is supplied then the properties are applied to individual borders as described here).

Valid property values (other than inherit and initial) are:

Value	Description
color	A CSS colour
`transparent`	(default value). Transparent

Default Value: The current CSS colour of the element
JavaScript syntax: e.g. *object*`.style.borderColor="red blue"`
Inherited: No
Animatable: Yes

border-image

[CSSPropertyBorderImage]

The CSS (CSS1) `border-image` property is a shorthand property combining (up to) 5 of the border-image properties. These allow you to specify that an image should be used instead of the normal border around an element.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- border-image-source
- border-image-slice
- border-image-width
- border-image-outset
- border-image-repeat

Default Value: `none 100% 1 0 stretch`
JavaScript syntax: e.g. *object*`.style.borderImage="url(myborder.png) 20 round"`
Inherited: No
Animatable: No

border-image-outset

[CSSPropertyBorderImageOutset]

The CSS (CSS3) `border-image-outset` property specifies the amount by which a border image area extends beyond the border box.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length that specifies how far from the edges the border-image will appear
number	Multiples of the relevant border-width

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.borderImageOutset="30px"`
Inherited: No
Animatable**:** No

border-image-repeat

[CSSPropertyBorderImageRepeat]

The CSS (CSS3) `border-image-repeat` property specifies whether and how the border image should be repeated, rounded or stretched.

Valid property values (other than inherit and initial) are:

Value	Description
`repeat`	Image tiled (i.e. repeated) to fill area
`round`	Image tiled (i.e. repeated) to fill area (with image rescaled so that it exactly fits the area)
`space`	Image tiled (i.e. repeated) to fill area (with extra space distributed around tiles if the fit is not exact
`stretch`	(default value). Image stretch to fill area

Default Value: `stretch`
JavaScript syntax: e.g. *object*`.style.borderImageRepeat="space"`
Inherited: No
Animatable**:** No

border-image-slice

[CSSPropertyBorderImageSlice]

The CSS (CSS3) `border-image-slice` property specifies how any border image should be sliced. The image is always sliced into nine sections (3 x 3, i.e. 4 corners, 4 edge non-corners and 1 middle). The middle part is fully transparent unless the fill keyword is set.

The property takes up to four values. If the fourth is omitted then it is given the same value as the second. If the third is omitted then it is given the same value as the first. If the second is also omitted then it is given the same value as the first.

Valid property values (other than inherit and initial) are:

Value	Description
number	Number(s) of pixels for raster images or coordinates for vector images
x%	Percentages relative to height or width of image
`fill`	Middle part of image is displayed

Default Value: `100%`
JavaScript syntax: e.g. *object*`.style.borderImageSlice="20%"`
Inherited: No
Animatable**:** No

border-image-source

[CSSPropertyBorderImageSource]

The CSS (CSS3) `border-image-source` property specifies path of image to be used as a border (instead of a normal border around an element).

Valid property values (other than inherit and initial) are:

Value	Description
`none`	(default value). No image used
url('URL')	URL path to the border image

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.borderImageSource="url(myborder.png)"`
Inherited: No
Animatable**:** No

border-image-width

[CSSPropertyBorderImageWidth]

The CSS (CSS3) `border-image-width` property specifies the width of a border image.

The property takes up to four values. If the fourth is omitted then it is given the same value as the second. If the third is omitted then it is given the same value as the first. If the second is also omitted then it is given the same value as the first.

Valid property values (other than inherit and initial) are:

Value	Description
length	CSS length specifying size of border-width
number	(default value). Multiples of corresponding border-width
%	Relative to size of border image area
`auto`	Intrinsic width or height of the corresponding image slice

Default Value: `1`
JavaScript syntax: e.g. *object*`.style.borderImageWidth="50px"`
Inherited: No
Animatable**:** No

border-left

[CSSPropertyBorderLeft]

The CSS (CSS1) `border-left` property is a shorthand property combining all the main left border properties.

Valid property values (other than inherit or initial) are defined by the elements of the shorthand and are:

- border-left-width
- border-left-style
- border-left-color

Missing properties are given their default values.

Default Value: `medium none` *color*
JavaScript syntax: e.g. *object*`.style.borderLeft="2px solid red"`
Inherited: No
Animatable**:** Yes

In the default value, *color* is the CSS colour of the element.

border-left-color

[CSSPropertyBorderLeftColor]

The CSS (CSS1) `border-left-color` property sets the colour of the left border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
color	Specified CSS colour
`transparent`	Transparent

Default Value: The current CSS colour of the element
JavaScript syntax: e.g. *object*`.style.borderLeftColor="red"`
Inherited: No
Animatable**:** Yes

border-left-style

[CSSPropertyBorderLeftStyle]

The CSS (CSS1) `border-left-style` property sets the border style of the left border of an element.

Valid property values (other than inherit and initial) are shown here.

Default Value:	`none`
JavaScript syntax:	e.g. *object*`.style.borderLeftStyle="groove"`
Inherited:	No
Animatable:	No

border-left-width

[CSSPropertyBorderLeftWidth]

The CSS (CSS1) `border-left-width` property sets the width of the left border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value:	`medium`
JavaScript syntax:	e.g. *object*`.style.borderLeftWidth="4px"`
Inherited:	No
Animatable:	Yes

border-radius

[CSSPropertyBorderRadius]

The CSS (CSS1) `border-radius` property is used to add rounded corners to an element. It is a shorthand way of setting the four individual border radius properties, see:

- border-top-left-radius
- border-top-right-radius
- border-bottom-right-radius
- border-bottom-left-radius

Valid property values (other than inherit and initial) are (single-value versions) of the valid properties for each of these individual elements.

Default Value:	`0`
JavaScript syntax:	e.g. *object*`.style.borderRadius="4px"`
Inherited:	No
Animatable:	Yes

Depending on the number of values supplied:

Number supplied	E.g.	Which values are applied to which corners
1	*x1*	*x1* applied to all four corners
2	*x1 x2*	*x1* applied to top-left and bottom-right corners *x2* applied to top-right and bottom-left corners
3	*x1 x2 x3*	*x1* applied to top-left corner *x2* applied to top-right and bottom-left corners *x3* applied to bottom-right corner
4	*x1 x2 x3 x4*	*x1* applied to top-left corner *x2* applied to top-right corner *x3* applied to bottom-right corner *x4* applied to bottom-left corner

border-right

[CSSPropertyBorderRight]

The CSS (CSS1) `border-right` property is a shorthand property combining all the main right border properties.

Valid property values (other than inherit or initial) are defined by the elements of the shorthand and are:

- border-right-width
- border-right-style
- border-right-color

Missing properties are given their default values.

Default Value: `medium none` *color*
JavaScript syntax: e.g. *object*`.style.borderRight="2px solid red"`
Inherited: No
Animatable**:** Yes

In the default value, *color* is the CSS colour of the element.

border-right-color

[CSSPropertyBorderRightColor]

The CSS (CSS1) `border-right-color` property sets the colour of the right border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
color	Specified CSS colour

`transparent`	Transparent

Default Value: The current CSS colour of the element
JavaScript syntax: e.g. *object*`.style.borderRightColor="red"`
Inherited: No
Animatable**:** Yes

border-right-style

[CSSPropertyBorderRightStyle]

The CSS (CSS1) `border-right-style` property sets the border style of the right border of an element.

Valid property values (other than inherit and initial) are shown here.

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.borderRightStyle="groove"`
Inherited: No
Animatable**:** No

border-right-width

[CSSPropertyBorderRightWidth]

The CSS (CSS1) `border-right-width` property sets the width of the right border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value: `medium`
JavaScript syntax: e.g. *object*`.style.borderRightWidth="4px"`
Inherited: No
Animatable**:** Yes

border-spacing

[CSSPropertyBorderSpacing]

The CSS (CSS2) `border-spacing` property sets the distance between borders of adjacent cells (if the border-collapse property is `separate` (which is its default).

Valid property values (other than inherit and initial) are:

Value	Description
length1 length2	Distance between borders of adjacent cells in CSS lengths. Must be non-negative. If one value supplied then both horizontal and vertical spacing. If two supplied then first relates to horizontal spacing and second to vertical spacing
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.borderSpacing="4px"`
Inherited: No
Animatable**:** Yes

border-style

[CSSPropertyBorderStyle]

The CSS (CSS1) `border-style` property sets the border style of an element's four borders. The individual border styles can be set separately using border-bottom-style, border-left-style, border-right-style and border-top-style. As with some other aggregate edge properties, up to four parameter values can be supplied (and if more than one is supplied then the properties are applied to individual borders as described here).

Valid property values (other than inherit and initial) are shown here.

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.borderStyle="double dashed"`
Inherited: No
Animatable**:** Yes

border-top

[CSSPropertyBorderTop]

The CSS (CSS1) `border-top` property is a shorthand property combining all the main top border properties.

Valid property values (other than inherit or initial) are defined by the elements of the shorthand and are:

- border-top-width
- border-top-style
- border-top-color

Missing properties are given their default values.

Default Value: `medium none` *color*
JavaScript syntax: e.g. *object*`.style.borderTop="2px solid red"`
Inherited: No

Animatable: Yes

In the default value, *color* is the CSS colour of the element.

border-top-color

[CSSPropertyBorderTopColor]

The CSS (CSS1) `border-top-color` property sets the colour of the top border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
color	Specified CSS colour
`transparent`	Transparent

Default Value: The current CSS colour of the element
JavaScript syntax: e.g. *object*`.style.borderTopColor="red"`
Inherited: No
Animatable: Yes

border-top-left-radius

[CSSPropertyBorderTopLeftRadius]

The CSS (CSS3) `border-top-left-radius` property determines the shape of the top-left corner of a border. It allows you to add rounded borders to elements.

The two length or percentage values define the radii of a quarter ellipse defining the shape of the corner. The first is the horizontal radius, the second the vertical radius. If either is omitted then it is copied from the other (so the corner is a quarter-circle). If either is zero then the corner becomes square. Percentages for the horizontal radius refer to the width of the border box, those for the vertical radius refer to the height of the border box.

Valid property values (other than inherit and initial) are:

Value	Description
length1, length2	Specified CSS lengths
%, %	See above

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.borderTopLeftRadius="5px 8px"`
Inherited: No
Animatable: Yes

border-top-right-radius

[CSSPropertyBorderTopRightRadius]

The CSS (CSS3) `border-top-right-radius` property determines the shape of the top-right corner of a border. It allows you to add rounded borders to elements.

The two length or percentage values define the radii of a quarter ellipse defining the shape of the corner. The first is the horizontal radius, the second the vertical radius. If either is omitted then it is copied from the other (so the corner is a quarter-circle). If either is zero then the corner becomes square. Percentages for the horizontal radius refer to the width of the border box, those for the vertical radius refer to the height of the border box.

Valid property values (other than inherit and initial) are:

Value	Description
length1, length2	Specified CSS lengths
%, %	See above

Default Value: *0*
JavaScript syntax: e.g. *object*`.style.borderTopRightRadius="5px 8px"`
Inherited: No
Animatable**:** Yes

border-top-style

[CSSPropertyBorderTopStyle]

The CSS (CSS1) `border-top-style` property sets the border style of the top border of an element.

Valid property values (other than inherit and initial) are shown here.

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.borderTopStyle="groove"`
Inherited: No
Animatable**:** No

border-top-width

[CSSPropertyBorderTopWidth]

The CSS (CSS1) `border-top-width` property sets the width of the top border of an element. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value: `medium`
JavaScript syntax: e.g. *object*`.style.borderTopWidth="4px"`
Inherited: No
Animatable**:** Yes

border-width

[CSSPropertyBorderWidth]

The CSS (CSS1) `border-width` property sets the width of an element's four borders. You should always specify the border-style property before this property, as an element must have a border before you can change its characteristics. The individual border widths can be set separately using border-bottom-width, border-left-width, border-right-width and border-top-width. As with some other aggregate edge properties, up to four parameter values can be supplied (and if more than one is supplied then the properties are applied to individual borders as described here).

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value: `medium`
JavaScript syntax: e.g. *object*`.style.borderWidth="10px 10px"`
Inherited: No
Animatable**:** Yes

bottom

[CSSPropertyBottom]

The CSS (CSS2) `bottom` property sets, for absolutely positioned elements, the bottom edge of an element relative to the corresponding edge of its nearest positioned ancestor. If such an element has no positioned ancestors then it uses the document body and moves along with page scrolling. A 'positioned' element is one whose position is anything other than static.

Valid property values (other than inherit and initial) are:

Value	Description
length	Sets edge position in CSS length units. Can be negative
%	Sets edge position in % of containing element. Can be negative
`auto`	(default value). Browser calculates edge position

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.bottom="10px"`
Inherited: No
Animatable**:** Yes

box-shadow

[CSSPropertyBoxShadow]

The CSS (CSS3) `box-shadow` property applies one or more shadows to an element. The property is a comma-separated list of shadows, each specified by 2 to 4 length values, an optional colour and an optional inset keyword. If a length is omitted then it is taken to be zero.

Valid property values (other than inherit and initial) are:

Value	Description
box-shadow parameter	See below
`none`	(default value). No shadow is displayed

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.boxShadow="5px 10px 15px red inset"`
Inherited: No
Animatable**:** Yes

The parameters of the box-shadow parameter are:

Value	Description
h-shadow	Position of horizontal shadow (can be negative)
v-shadow	Position of vertical shadow (can be negative)
blur	Optional. Blur distance
spread	Optional. Size of shadow (can be negative)
color	Optional (for most browsers). Shadow CSS colour. Default value is black.
`inset`	Optional. Changes shadow from an outer shadow to an inner shadow

box-sizing

[CSSPropertyBoxSizing]

The CSS (CSS3) `box-sizing` property indicates what the sizing properties of a box (width and height) should be applied to, i.e. just the content-box or some broader definition.

Valid property values (other than inherit and initial) are:

Value	Description
`border-box`	Height and width properties (and min-height / min-width / max-height / max-width properties) include content, border and padding (but not margin)
`content-box`	(default value). Height and width properties (and min-height / min-width / max-height / max-width properties) include only the content, and do not include border, padding or margin

Default Value: `content-box`

JavaScript syntax: e.g. *object*`.style.boxSizing="border-box"`
Inherited: No
Animatable: No

caption-side

[CSSPropertyCaptionSide]

The CSS (CSS2) `caption-side` property indicates where a table caption should be placed.

Valid property values (other than inherit and initial) are:

Value	Description
`bottom`	Puts caption below table
`top`	(default value). Puts caption above table

Default Value: `top`
JavaScript syntax: e.g. *object*`.style.captionSide="bottom"`
Inherited: Yes
Animatable: No

clear

[CSSPropertyClear]

The CSS (CSS1) `clear` property indicates on which sides of an element a floating element is not allowed to float.

Valid property values (other than inherit and initial) are:

Value	Description
`both`	Floating elements not allowed on either left or right side
`left`	Floating elements not allowed on left side
`none`	(default value). Allows floating elements on both sides
`right`	Floating elements not allowed on right side

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.clear="both"`
Inherited: No
Animatable: No

clip

[CSSPropertyClip]

The CSS (CSS2) `clip` property indicates what to do with an image that is larger than its containing element. The clip property allows you to specify a rectangle to clip an absolutely positioned element. The rectangle is specified by four coordinates, all from the top-left corner of the element being clipped. The *clip* property does not work if the overflow property is set to visible.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). No clipping is applied
shape	Clips an element to within a given shape

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.clip="rect(0px,10px,10px,5px)"`
Inherited: No
Animatable**:** Yes

At the time of writing, the only valid type of value for *shape* was `rect`(*top*, *right*, *bottom*, *left*) where *top*, *right*, *bottom* and *left* are CSS lengths.

color

[CSSPropertyColor]

The CSS (CSS1) `color` property indicates the colour of text within an element. Usually you should choose a combination of background colour and text colour that makes the text easy to read.

Valid property values (other than inherit and initial) are:

Value	Description
color	A CSS colour

Default Value: N/A
JavaScript syntax: e.g. *object*`.style.color="red"`
Inherited: Yes
Animatable**:** Yes

column-count

[CSSPropertyColumnCount]

The CSS (CSS3) `column-count` property specifies the number of columns an element (e.g. a paragraph) should be divided into.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Number of columns will be determined by other properties such as column-width
number	Optimum number of columns into which content of element will be formatted

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.columnCount=2`
Inherited: No
Animatable**:** Yes

column-fill

[CSSPropertyColumnFill]

The CSS (CSS3) `column-fill` property specifies how to fill columns.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	Columns filled sequentially and therefore may have different lengths
`balance`	(default value). Columns are balanced and browsers should minimise variation in column length

Default Value: `balance`
JavaScript syntax: e.g. *object*`.style.columnFill="auto"`
Inherited: No
Animatable**:** No

column-gap

[CSSPropertyColumnGap]

The CSS (CSS3) `column-gap` property specifies what gap is placed between columns of an element. Any column rule between columns will appear in the middle of the gap.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
`normal`	(default value). Specifies normal gap between columns (W3C suggests 1em)

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.columnGap="20px"`
Inherited: No
Animatable**:** Yes

column-rule

[CSSPropertyColumnRule]

The CSS (CSS1) `column-rule` property is a shorthand property combining (up to) 3 of the column-rule properties.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- column-rule-width
- column-rule-style

- column-rule-color

Default Value: *See individual properties*
JavaScript syntax: e.g. *object*`.style.columnRule="3px outset red"`
Inherited: No
Animatable**:** *See individual properties*

column-rule-color

[CSSPropertyColumnRuleColor]

The CSS (CSS3) `column-rule-color` property specifies the colour of any rule between columns.

Valid property values (other than inherit and initial) are:

Value	Description
color	A CSS colour

Default Value: *Current colour of element*
JavaScript syntax: e.g. *object*`.style.columnRuleColor="red"`
Inherited: No
Animatable**:** Yes

column-rule-style

[CSSPropertyColumnRuleStyle]

The CSS (CSS3) `column-rule-style` property specifies the style of any rule between columns.

Valid property values (other than inherit and initial) are:

Value	Description
`dashed`	Dashed border
`dotted`	Dotted border
`double`	Double border
`groove`	Effect depends on column-rule-width and column-rule-color values
`hidden`	Hidden
`inset`	Effect depends on column-rule-width and column-rule-color values
`none`	(default value). No border
`outset`	Effect depends on column-rule-width and column-rule-color values
`ridge`	Effect depends on column-rule-width and column-rule-color values
`solid`	Solid border

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.columnRuleStyle="dotted"`
Inherited: No
Animatable**:** No

column-rule-width

[CSSPropertyColumnRuleWidth]

The CSS (CSS3) `column-rule-width` property specifies the width of any rule between columns.

Valid property values (other than inherit and initial) are:

Value	Description
length	Specified thickness as a CSS length
`medium`	(default value). Medium width
`thick`	Thick width
`thin`	Thin width

Default Value: `medium`
JavaScript syntax: e.g. *object*`.style.columnRuleWidth="5px"`
Inherited: No
Animatable**:** Yes

columns

[CSSPropertyColumns]

The CSS (CSS1) `columns` property is a shorthand property for setting certain column properties for an element.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- column-width
- column-count

Default Value: `auto auto`
JavaScript syntax: e.g. *object*`.style.columns="80px 3"`
Inherited: No
Animatable**:** *See individual properties*

column-span

[CSSPropertyColumnSpan]

The CSS (CSS3) `column-span` property specifies how many columns an element should span across.

Valid property values (other than inherit and initial) are:

Value	Description
`1`	(default value). Element should span across one column
`all`	Element should span across all columns

Default Value: `1`
JavaScript syntax: e.g. *object*`.style.columnSpan="all"`

Inherited: No
Animatable: No

column-width

[CSSPropertyColumnWidth]

The CSS (CSS3) `column-width` property provides a suggested optimal width for columns.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
`auto`	(default value). Column width will be determined by browser

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.columnWidth="200px"`
Inherited: No
Animatable: Yes

cursor

[CSSPropertyCursor]

The CSS (CSS2) `cursor` property indicates the type of cursor to be displayed when pointing to an element. Usually this property is set in a manner that conventionally indicates what is likely to happen next (if the cursor is clicked).

Valid property values (other than inherit and initial) are:

Value	Description
`alias`	Indicates alias is to be created
`all-scroll`	Indicates that something can be scrolled in any direction
`auto`	(default value). Browser selects cursor format
`cell`	Indicates a cell (or set of cells) can be selected
`context-menu`	Indicates a context-menu is available
`col-resize`	Indicates column can be resized horizontally
`copy`	Indicates something to be copied
`crosshair`	Cursor appears as a crosshair
`default`	Cursor appears as the default cursor
`e-resize`	Indicates edge of box can be moved right (i.e. 'east')
`ew-resize`	Indicates a bidirectional resize cursor
`grab`	Indicates something can be grabbed
`grabbing`	Indicates something can be grabbed
`help`	Indicates help is available
`move`	Indicates can move something
`n-resize`	Indicates edge of box can be moved up (i.e. 'north')
`ne-resize`	Indicates edge of a box can be moved up and right (i.e. 'north-east')
`nesw-resize`	Indicates a bidirectional resize cursor
`ns-resize`	Indicates a bidirectional resize cursor

nw-resize	Indicates edge of a box can be moved up and left (i.e. 'north-west')
nwse-resize	Indicates a bidirectional resize cursor
no-drop	Indicates can't drop dragged item here
none	No cursor shown
not-allowed	Indicates requested action won't be executed
pointer	Indicates a link
progress	Indicates program is busy
row-resize	Indicates row can be resized vertically
s-resize	Indicates edge of box can be moved down (i.e. 'south')
se-resize	Indicates edge of a box can be moved down and right (i.e. 'south-east')
sw-resize	Indicates edge of a box can be moved down and left (i.e. 'south-west')
text	Indicates text may be selected
URL	A comma separated list of URLs pointing to custom cursors. You should ideally specify a generic cursor at the end of the list (in case none of the URL-defined cursors can be used)
vertical-text	Indicates vertical-text can be selected
w-resize	Indicates edge of box can be moved left (i.e. 'west')
wait	Indicates program is busy
zoom-in	Indicates something can be zoomed in
zoom-out	Indicates something can be zoomed out

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.cursor="zoom-in"`
Inherited: Yes
Animatable**:** No

direction

[CSSPropertyDirection]

The CSS (CSS2) `direction` property specifies the text or writing direction. It can be used with the unicode-bidi property to set or indicate whether text should be overridden to support multiple languages (that are formatted in different directions) in the same document.

Valid property values (other than inherit and initial) are:

Value	Description
ltr	(default value). Text direction is left-to-right
rtl	Text direction is right-to-left
zoom-in	Indicates something can be zoomed in
zoom-out	Indicates something can be zoomed out

Default Value: `ltr`
JavaScript syntax: e.g. *object*`.style.direction="rtl"`
Inherited: Yes
Animatable**:** No

display

[CSSPropertyDisplay]

The CSS (CSS1 and some values that are new in CSS3) `display` property indicates the type of box to be used for an element.

Valid property values (other than inherit and initial) are:

Value	Description
`block`	Displayed as a block element (like default <p> element)
`flex`	Displayed as a block-level flex container
`inline`	(default value). Displayed in-line (like default element)
`inline-block`	Displayed as an in-line block container (i.e. the inside is formatted like a block-level box but element itself as an inline-level box)
`inline-flex`	Displayed as an inline-level flex container
`inline-table`	Displayed as an inline-level table
`list-item`	Displayed like a default element
`none`	Not displayed
`run-in`	Displayed as either block or inline depending on context
`table`	Displayed like a default <table> element
`table-caption`	Displayed like a default <caption> element
`table-cell`	Displayed like a default <td> element
`table-column`	Displayed like a default <col> element
`table-column-group`	Displayed like a default <colgroup> element
`table-footer-group`	Displayed like a default <tfoot> element
`table-header-group`	Displayed like a default <thead> element
`table-row`	Displayed like a default <tr> element
`table-row-group`	Displayed like a default <tbody> element

Default Value: `inline`
JavaScript syntax: e.g. *object*`.style.display="none"`
Inherited: No
Animatable**:** No

empty-cells

[CSSPropertyEmptyCells]

The CSS (CSS2) `empty-cells` property indicates whether to display borders and background for empty cells in a table (only applicable if border-collapse property is "separate").

Valid property values (other than inherit and initial) are:

Value	Description
`hide`	No background or borders shown on empty cells
`show`	(default value). Background and borders shown on empty cells

Default Value: `show`
JavaScript syntax: e.g. *object*`.style.emptyCells="hide"`
Inherited: Yes

Animatable: No

filter

[CSSPropertyFilter]

The CSS (CSS3) `filter` property applies visual effects like grayscale, blur and saturation to an element (usually an <img> element).

Valid property values (other than inherit and initial) are:

Value	Description
`blur` (*length*)	Blur added. Length needs to be in px, e.g. `5px`. Larger value creates more blur. If no value is specified then `0` is used which leaves image unaltered
`brightness` (*%* or *number*)	Brightness adjusted. `0%` (or `0`) makes completely black, `100%` (or `1`) leaves image unaltered and is the default. Values over `100%` will add to brightness.
`contrast` (*%* or *number*)	Contrast adjusted. `0%` (or `0`) makes completely black, `100%` (or `1`) leaves image unaltered and is the default. Values over `100%` will add to contrast.
`drop-shadow` (*h-shadow v-shadow blur spread color*)	Drop shadow applied: *h-shadow* (required). Value (in px) for horizontal shadow (negative values have shadow on left of image *v-shadow* (required). Value (in px) for horizontal shadow (negative values have shadow above image *blur* (optional). Value (in px) adding blur effect to shadow, see above *spread* (optional). Value (in px) which causes shadow to expand (positive) or shrink (negative). Some browsers do not support this parameter *color* (optional). Adds CSS colour to the shadow, default depends on browser but is usually black.
`grayscale` (*%* or *number*)	Converts to grayscale. `0%` (or `0`) is default and leaves image unaltered, `100%` (or `1`) makes image completely gray (i.e. black and white).
`hue-rotate` (*angle*)	Applies a hue rotation to image (see CSS colours for more details of hues). Angle needs to be in degrees, e.g. `20deg`. Default is `0deg`, which leaves image unaltered
`invert` (*%* or *number*)	Inverts colours/brightness in image. `0%` (or `0`) is default and leaves image unaltered, `100%` (or `1`) completely inverts colouring.
`none`	(default value). No effect applied
`opacity` (% or *number*)	Sets opacity level (i.e. degree of transparency) for image. `0%` (or `0`) is completely transparent. `100%` (or `1`) is default and leaves image unaltered (no transparency). Is similar to opacity property.
`saturate` (% or *number*)	Saturates image. `0%` (or `0`) makes image completely unsaturated. 100% (or 1) is default and leaves image unaltered. Values over `100%` (or `1`) add saturation.
`sepia` (% or *number*)	Converts image to sepia. `0%` (or `0`) is default and leaves image unaltered, 100% (or 1) makes image completely sepia (i.e. old photography style black and white).

`url(`*filename*`)`	Takes the location of an XML file that specifies an SVG filter (including potentially an anchor to a specific filter element, e.g.: `filter: url(`*svg-url#svg-element-id*`)`

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.filter="grayscale(90%)"`
Inherited: No
Animatable**:** Yes

flex

[CSSPropertyFlex]

The CSS (CSS3) `flex` property is a shorthand property for setting certain flex properties for an element.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- flex-grow
- flex-shrink
- flex-basis

Default Value: `0 1 auto`
JavaScript syntax: e.g. *object*`.style.flex="1"`
Inherited: No
Animatable**:** *See individual properties*

flex-basis

[CSSPropertyFlexBasis]

The CSS (CSS3) `flex-basis` property indicates the initial length of a flexible element. If an element is not a flexible element then this property has no effect.

Valid property values (other than inherit and initial) are:

Value	Description
number	A length unit or percentage specifying initial length of flexible item
`auto`	(default value). Length is equal to length of flexible item or if not specified is set according to context

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.flexBasis="100px"`
Inherited: No
Animatable**:** Yes

flex-direction

[CSSPropertyFlexDirection]

The CSS (CSS3) `flex-direction` property indicates the direction of a flexible element. If an element is not a flexible element then this property has no effect.

Valid property values (other than inherit and initial) are:

Value	Description
`column`	Items are displayed vertically, as a column
`column-reverse`	Items are displayed vertically, but in reverse order
`row`	(default value). Items are displayed horizontally, as a row
`row-reverse`	Items are displayed horizontally, but in reverse order

Default Value: `row`
JavaScript syntax: e.g. *object*`.style.flexDirection="column"`
Inherited: No
Animatable**:** No

flex-flow

[CSSPropertyFlexFlow]

The CSS (CSS3) `flex-flow` property is a shorthand property for setting certain flex properties for a flex element.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- flex-direction
- flex-wrap

Default Value: `row nowrap`
JavaScript syntax: e.g. *object*`.style.flexFlow="column nowrap"`
Inherited: No
Animatable**:** *See individual properties*

flex-grow

[CSSPropertyFlexGrow]

The CSS (CSS3) `flex-grow` property indicates how much an element will grow relative to the rest of the flexible items in a container. If an element is not a flexible element then this property has no effect.

Valid property values (other than inherit and initial) are:

Value	Description
number	Specifies how much an item will grow relative to rest of flexible items

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.flexGrow="4"`

Inherited: No
Animatable: Yes

flex-shrink

[CSSPropertyFlexShrink]

The CSS (CSS3) `flex-shrink` property indicates how much an element will shrink relative to the rest of the flexible items in a container. If an element is not a flexible element then this property has no effect. Note: for some browsers, the resulting sizes of elements can be less intuitive than using the apparently analogous fractional value for the flex-grow property.

Valid property values (other than inherit and initial) are:

Value	Description
number	Specifies how much an item will shrink relative to rest of flexible items

Default Value: `1`
JavaScript syntax: e.g. *object*`.style.flexShrink="4"`
Inherited: No
Animatable: Yes

flex-wrap

[CSSPropertyFlexWrap]

The CSS (CSS3) `flex-wrap` property indicates whether flexible items should wrap. If an element is not a flexible element then this property has no effect.

Valid property values (other than inherit and initial) are:

Value	Description
`nowrap`	(default value). Will not wrap
`wrap`	Will wrap if necessary
`wrap-reverse`	Will wrap if necessary, but in reverse order

Default Value: `nowrap`
JavaScript syntax: e.g. *object*`.style.flexWrap="wrap-reverse"`
Inherited: No
Animatable: No

float

[CSSPropertyFloat]

The CSS (CSS1) `float` property indicates whether an element should float. If the element is absolutely positioned then the float property is ignored.

Valid property values (other than inherit and initial) are:

Value	Description
`left`	Element floats to left
`none`	(default value). Element not floated (i.e. displays exactly where it occurs in text)
`right`	Element floats to right

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.cssFloat="right"`
Inherited: No
Animatable**:** No

font

[CSSPropertyFont]

The CSS (CSS1) `font` property is a shorthand property for setting the font properties of an element.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- font-style
- font-variant
- font-weight
- font-size i.e. line height (required)
- font-family (required)

Default Value: *See individual properties*
JavaScript syntax: e.g. *object*`.style.font="italic 12px arial"`
Inherited: Yes
Animatable**:** *See individual properties*

font-family

[CSSPropertyFontFamily]

The CSS (CSS1) `font-family` property indicates the font to be used for an element. It can include several values, separated by commas (typically starting from more specific fonts and ending with more generic fonts). If the browser doesn't support the first font in the list, it tries the next one etc. Font family names can be:

- Specific, e.g. `"Times New Roman"`, `Arial`; or
- Generic, e.g. `serif`, `cursive`

Valid property values (other than inherit and initial) are:

Value	Description
family-name / generic-family / list	A prioritised list of specific or generic font family names. Note: if a font name contains a space then it must be enclosed in quotes

Default Value: *Depends on browser*
JavaScript syntax: e.g. *object*`.style.fontFamily="Verdana,serif"`
Inherited: Yes
Animatable**:** No

font-size

[CSSPropertyFontSize]

The CSS (CSS1) `font-size` property indicates the size of the font to be used for an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Fixed size as a CSS length (px, cm, …)
%	Set as a percentage of parent element's font size
`large`	Large size
`larger`	Larger size than parent element's font size
`medium`	(default value). Medium size
`small`	Small size
`smaller`	Smaller size than parent element's font size
`x-large`	X-large size
`xx-large`	XX-large size
`x-small`	X-small size
`xx-small`	XX-small size

Default Value: `medium`
JavaScript syntax: e.g. *object*`.style.fontSize="12px"`
Inherited: Yes
Animatable**:** Yes

font-size-adjust

[CSSPropertyFontSizeAdjust]

The CSS (CSS3) `font-size-adjust` property gives better control of the font size than is provided by the font-size property alone, when the first font selected in the font-family property is not available. All fonts have an "aspect value" which is the size-difference between the lowercase "x" and the uppercase "X". If the browser is told this value then it can figure out what font-size to use when displaying text from the entry in the font-family property that is actually used.

Valid property values (other than inherit and initial) are:

Value	Description
number	A value indicating the aspect value to use
`none`	(default value). No adjustment applied to font size

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.fontSizeAdjust="0.58"`
Inherited: Yes

Animatable: Yes

font-stretch

[CSSPropertyFontStretch]

The CSS (CSS3) `font-stretch` property makes text in an element narrower or more stretched out than usual.

Valid property values (other than inherit and initial) are:

Value	Description
`condensed`	Narrower than semi-condensed
`expanded`	Wider than semi-expanded
`extra-condensed`	Narrower than condensed
`extra-expanded`	Wider than expanded
`normal`	(default value). No adjustment applied
`semi-condensed`	Narrower than normal
`semi-expanded`	Wider than normal
`ultra-condensed`	Narrower than extra-condensed
`ultra-expanded`	Wider than extra-expanded

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.fontStretch="condensed"`
Inherited: Yes
Animatable: Yes

font-style

[CSSPropertyFontStyle]

The CSS (CSS1) `font-style` property indicates the font style to use for text in an element.

Valid property values (other than inherit and initial) are:

Value	Description
`italic`	Italic text
`normal`	(default value). Normal text
`oblique`	Oblique text

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.fontStyle="italic"`
Inherited: Yes
Animatable: No

font-variant

[CSSPropertyFontVariant]

The CSS (CSS1) `font-variant` property indicates whether text should be in a small-caps font (in which all lowercase letters are converted to slightly smaller uppercase letters).

Valid property values (other than inherit and initial) are:

Value	Description
`normal`	(default value). Normal text
`small-caps`	Text is shown in a small-caps font

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.fontVariant="small-caps"`
Inherited: Yes
Animatable**:** No

font-weight

[CSSPropertyFontWeight]

The CSS (CSS1) `font-weight` property indicates how thick or thin (i.e. bold or not) characters should be in the text of an element.

Valid property values (other than inherit and initial) are:

Value	Description
`100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900`	Number selects from very thin to very thick (400 is normal, 700 is bold)
`bold`	Bold text
`bolder`	Very bold text
`lighter`	Lighter text
`normal`	(default value). Normal text

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.fontWeight="bold"`
Inherited: Yes
Animatable**:** Yes

hanging-punctuation

[CSSPropertyHangingPunctuation]

The CSS (CSS3) `hanging-punctuation` property indicates whether a punctuation mark can be placed outside the box at the start or end of a full line of text. At the time of writing many major browsers did not support this property.

Valid property values (other than inherit and initial) are:

Value	Description
`allow-end`	Punctuation may hang outside end edge of all lines
`first`	May hang outside start edge
`force-end`	May hang outside end edge of all lines (and will be forced to do so if

	justification applies to the line)
`last`	May hang outside end edge
`none`	(default value). Punctuation mark cannot be placed outside line box

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.hangingPunctuation="last"`
Inherited: Yes
Animatable**:** No

height

[CSSPropertyHeight]

The CSS (CSS1) `height` property indicates the height of an element (excluding padding, borders and margins). It is overridden by the min-height or max-height properties, if either of them are present.

Valid property values (other than inherit and initial) are:

Value	Description
length	Height as a CSS length (px, cm, …)
%	Height of element defined as a percentage of height of its containing block
`auto`	(default value). Browser determines height

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.height="200px"`
Inherited: No
Animatable**:** Yes

justify-content

[CSSPropertyJustifyContent]

The CSS (CSS3) `justify-content` property indicates how to align a flexible container's items when the items do not use all available space along the horizontal axis.

Valid property values (other than inherit and initial) are:

Value	Description
`center`	Items are centred
`flex-start`	(default value). Items positioned at start of container
`flex-end`	Items positioned at end of container
`space-around`	Items positioned with space before, between and after lines
`space-between`	Items positioned with space between lines

Default Value: `flex-start`
JavaScript syntax: e.g. *object*`.style.justifyContent="space-around"`
Inherited: No
Animatable**:** Yes

left

[CSSPropertyLeft]

The CSS (CSS2) `left` property sets, for absolutely positioned elements, the left edge of an element relative to the corresponding edge of its nearest positioned ancestor. If such an element has no positioned ancestors then it uses the document body and moves along with the page scrolling. A 'positioned' element is one whose position is anything other than static.

Valid property values (other than inherit and initial) are:

Value	Description
length	Sets edge position as a CSS length. Can be negative
%	Sets edge position in % of containing element. Can be negative
`auto`	(default value). Browser calculates edge position

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.left="10px"`
Inherited: No
Animatable**:** Yes

letter-spacing

[CSSPropertyLetterSpacing]

The CSS (CSS1) `letter-spacing` property identifies the amount of space between consecutive text characters.

Valid property values (other than inherit and initial) are:

Value	Description
length	Amount of extra space as a CSS length between characters. Can be negative
`normal`	(default value). No extra space between characters

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.letterSpacing="4px"`
Inherited: Yes
Animatable**:** Yes

line-height

[CSSPropertyLineHeight]

The CSS (CSS1) `line-height` property identifies the height of lines of text.

Valid property values (other than inherit and initial) are:

Value	Description

length	Line height is a fixed height defined by this CSS length
number	Line height set as a multiple of the current font size
%	Line height set as a percentage of the current font size
`normal`	(default value). Line height is normal (given the relevant font size)

Default Value: normal
JavaScript syntax: e.g. *object*`.style.lineHeight="15px"`
Inherited: Yes
Animatable**:** Yes

list-style

[CSSPropertyListStyle]

The CSS (CSS1) `list-style` property is a shorthand property combining up to 3 list properties.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear) are:

- list-style-type
- list-style-position
- list-style-image

Default Value: `disc outside none`
JavaScript syntax: e.g. *object*.style`.listStyle="decimal inside"`
Inherited: Yes
Animatable**:** See *individual properties*

list-style-image

[CSSPropertyListStyleImage]

The CSS (CSS1) `list-style-image` property identifies an image that should be used as the list-item marker.

Valid property values (other than inherit and initial) are:

Value	Description
`none`	(default value). No image used. Instead, the type of list marker used is defined by the list-style-type property
url	URL path defining the image to be used as the list-item marker

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.listStyleImage="url('picture.gif')"`
Inherited: Yes
Animatable**:** No

list-style-position

[CSSPropertyListStylePosition]

The CSS (CSS1) `list-style-position` property identifies whether a list marker (e.g. a bullet character or (a), (b), (c) etc.) is inside or outside the relevant content container.

Valid property values (other than inherit and initial) are:

Value	Description
`inside`	Marker and text are indented and appear inside the relevant content container
`outside`	(default value). Marker is kept outside the relevant content container

Default Value: `outside`
JavaScript syntax: e.g. *object*`.style.listStylePosition="inside"`
Inherited: Yes
Animatable**:** No

list-style-type

[CSSPropertyListStyleType]

The CSS (CSS1) `list-style-type` property identifies the type of list-item marker used for a specified list.

Valid property values (other than inherit and initial) are:

Value	Description
`armenian`	Armenian numbering
`circle`	A circle
`cjk-ideographic`	Plain ideographic numbers
`decimal`	A number
`decimal-leading-zero`	A number with leading zeros (i.e. 01, 02, … if reaches beyond 9 but not beyond 99
`disc`	(default value). A filled circle
`georgian`	Georgian numbering
`hebrew`	Hebrew numbering
`hiragana`	Hiragana numbering
`hiragana-iroha`	Hiragana iroha numbering
`katakana`	Katakana numbering
`katakana-iroha`	Katakana iroha numbering
`lower-alpha`	I.e. a, b, c, …
`lower-greek`	I.e. lower case greek, α, β, γ, …
`lower-latin`	I.e. a, b, c, …
`lower-roman`	I.e. i, ii, iii, …
`none`	No marker
`square`	Square marker
`upper-alpha`	I.e. A, B, C, …
`upper-latin`	I.e. A, B, C, …
`upper-roman`	I.e. I, II, III, …

Default Value: `disc`
JavaScript syntax: e.g. *object*`.style.listStyleType="decimal"`
Inherited: Yes
Animatable**:** No

margin

[CSSPropertyMargin]

The CSS (CSS1) `margin` property is a shorthand property combining all four margin properties. The individual margin widths can be set separately using margin-bottom, margin-left, margin-right and margin-top. As with some other aggregate edge properties, up to four parameter values can be supplied (and if more than one is supplied then the properties are applied to individual borders as described here).

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements included in the *margin* property are:

Value	Description
length	Any specified margin size as a CSS length
%	Margin specified as percentage of width of container
`auto`	Browser calculates margin

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.margin="25px 30px"`
Inherited: No
Animatable**:** See *individual properties*

margin-bottom

[CSSPropertyMarginBottom]

The CSS (CSS1) `margin-bottom` property sets the width of the bottom margin of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified margin size as a CSS length
%	Margin specified as percentage of width of container
`auto`	Browser calculates margin

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.marginBottom="30px"`
Inherited: No
Animatable**:** Yes

margin-left

[CSSPropertyMarginLeft]

The CSS (CSS1) `margin-left` property sets the width of the left margin of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified margin size as a CSS length
%	Margin specified as percentage of width of container
`auto`	Browser calculates margin

Default Value: 0
JavaScript syntax: e.g. *object*`.style.marginLeft="30px"`
Inherited: No
Animatable**:** Yes

margin-right

[CSSPropertyMarginRight]

The CSS (CSS1) `margin-right` property sets the width of the right margin of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified margin size as a CSS length
%	Margin specified as percentage of width of container
`auto`	Browser calculates margin

Default Value: 0
JavaScript syntax: e.g. *object*`.style.marginRight="30px"`
Inherited: No
Animatable**:** Yes

margin-top

[CSSPropertyMarginTop]

The CSS (CSS1) `margin-top` property sets the width of the top margin of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified margin size as a CSS length
%	Margin specified as percentage of width of container
`auto`	Browser calculates margin

Default Value: 0
JavaScript syntax: e.g. *object*`.style.marginTop="30px"`
Inherited: No
Animatable**:** Yes

max-height

[CSSPropertyMaxHeight]

The CSS (CSS2) `max-height` property sets the maximum height an element can become. It overrides the height property.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
%	Defined as a percentage of that of the containing block
`none`	(default value). No limit

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.maxHeight="200px"`
Inherited: No
Animatable**:** Yes

max-width

[CSSPropertyMaxWidth]

The CSS (CSS2) `max-width` property sets the maximum width an element can become. It overrides the width property.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
%	Defined as a percentage of that of the containing block
`none`	(default value). No limit

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.maxWidth="300px"`
Inherited: No
Animatable**:** Yes

min-height

[CSSPropertyMinHeight]

The CSS (CSS2) `min-height` property sets the minimum height an element can become. It overrides the height property and the max-height property.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
%	Defined as a percentage of that of the containing block

none	(default value). No limit

Default Value: none
JavaScript syntax: e.g. *object*.style.minHeight="100px"
Inherited: No
Animatable**:** Yes

min-width

[CSSPropertyMinWidth]

The CSS (CSS2) min-width property sets the minimum width an element can become. It overrides the width property and the max-width property.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
%	Defined as a percentage of that of the containing block
none	(default value). No limit

Default Value: none
JavaScript syntax: e.g. *object*.style.minWidth="150px"
Inherited: No
Animatable**:** Yes

nav-down, nav-index, nav-left, nav-right, nav-up

[CSSPropertyNav]

At the time of writing the CSS (CSS3) nav-down, nav-index, nav-left, nav-right and nav-up properties appear to be supported by very few browsers, so are not explained further here. They indicate where to navigate to when using the down arrow, left arrow, right arrow and up arrow keys respectively.

The nav-index property specifies the sequential navigation order (i.e. the 'tabbing order') for an element.

opacity

[CSSPropertyOpacity]

The CSS (CSS3) opacity property sets the degree of opacity (transparency) of an element. 0 is completely transparent, 1 is completely non-transparent. Note also sets the transparency of all relevant child elements (if you don't want this to happen then use RGBA colouring).

Valid property values (other than inherit and initial) are:

Value	Description
number	From 0.0 to 1.0

Default Value: 1
JavaScript syntax: e.g. *object*`.style.opacity="0.6"`
Inherited: No
Animatable**:** Yes

order

[CSSPropertyOrder]

The CSS (CSS3) `order` property indicates the order of a flexible item relative to other flexible items inside the same container.

Valid property values (other than inherit and initial) are:

Value	Description
number	Integer

Default Value: 0
JavaScript syntax: e.g. *object*`.style.order="2"`
Inherited: No
Animatable**:** Yes

orphans

[CSSPropertyOrphans]

The CSS (CSS3) `orphans` property indicates the minimum number of lines of a paragraph that can be left on an old page. It works primarily with paged media, in which content is split into pages.

Valid property values (other than inherit and initial) are:

Value	Description
number	Integer

Default Value: 2
JavaScript syntax: e.g. *object*`.style.orphans="3"`
Inherited: No
Animatable**:** Yes

outline

[CSSPropertyOutline]

The CSS (CSS2) `outline` property is a shorthand property combining (up to) 3 of the outline properties.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- outline-color
- outline-style
- outline-width

Default Value: `invert none medium`
JavaScript syntax: e.g. *object*`.style.outline="2px solid red"`
Inherited: No
Animatable**:** See *individual properties*

outline-color

[CSSPropertyOutlineColor]

The CSS (CSS2) `outline-color` property sets the color of the outline of an element (i.e. a line that is drawn around the element, outside its borders, usually to make the element stand out relative to other elements.

Valid property values (other than inherit and initial) are:

Value	Description
color	A CSS colour
`invert`	(default value). Inverts colour

Default Value: `invert`
JavaScript syntax: e.g. *object*`.style.outlineColor="red"`
Inherited: No
Animatable**:** Yes

outline-offset

[CSSPropertyOutlineOffset]

The CSS (CSS2) `outline-offset` property sets the amount of space between an element's outline and the edge or border of the element.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.outlineOffset="10px"`
Inherited: No
Animatable**:** Yes

outline-style

[CSSPropertyOutlineStyle]

The CSS (CSS2) `outline-style` property specifies the style to be used for the outline of an element.

Valid property values (other than inherit and initial) are:

Value	Description
`dashed`	Dashed outline
`dotted`	Dotted outline
`double`	Double outline
`groove`	Effect depends on outline-color value
`hidden`	Hidden outline
`inset`	Effect depends on outline-color value
`none`	(default value). No outline
`outset`	Effect depends on outline-color value
`ridge`	Effect depends on outline-color value
`solid`	Solid outline

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.outlineStyle="ridge"`
Inherited: No
Animatable**:** Yes

outline-width

[CSSPropertyOutlineWidth]

The CSS (CSS2) `outline-width` property sets the width of an element's outline (outside the edge or border of the element). Note: an element needs to have an outline (so you need to set the outline-style property to something other than none) before the width of the outline can be set.

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.outlineWidth="3px"`
Inherited: No
Animatable**:** Yes

overflow

[CSSPropertyOverflow]

The CSS (CSS2) `overflow` property indicates what happens when content overflows an element's box.

Valid property values (other than inherit and initial) are:

Value	Description

`auto`	Overflow is clipped and a scroll-bar should typically be added
`hidden`	Overflow is clipped. No scroll-bar is added, so rest of content is effectively invisible
`scroll`	Overflow is clipped and a scroll-bar is added
`visible`	(default value). Overflow is not clipped and is rendered outside the element's box

Default Value: `visible`
JavaScript syntax: e.g. *object*`.style.overflow="scroll"`
Inherited: No
Animatable**:** No

overflow-x

[CSSPropertyOverflowX]

The CSS (CSS2) `overflow-x` property indicates what to with left/right edges of content overflowing an element's box.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	Overflow is clipped and a scroll-bar should typically be added
`hidden`	Overflow is clipped. No scroll-bar is added, so rest of content is effectively invisible
`scroll`	Overflow is clipped and a scroll-bar is added
`visible`	(default value). Overflow is not clipped and may be rendered outside the element's box

Default Value: `visible`
JavaScript syntax: e.g. *object*`.style.overflowX="scroll"`
Inherited: No
Animatable**:** No

overflow-y

[CSSPropertyOverflowY]

The CSS (CSS2) `overflow-y` property indicates what to with top/bottom edges of content overflowing an element's box.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	Overflow is clipped and a scroll-bar should typically be added
`hidden`	Overflow is clipped. No scroll-bar is added, so rest of content is effectively invisible
`scroll`	Overflow is clipped and a scroll-bar is added
`visible`	(default value). Overflow is not clipped and may be rendered outside the element's box

Default Value: `visible`
JavaScript syntax: e.g. *object*`.style.overflowY="scroll"`
Inherited: No
Animatable**:** No

padding

[CSSPropertyPadding]

The CSS (CSS1) `padding` property is a shorthand property combining the 4 padding sub-properties. The individual padding widths can be set separately using padding-bottom, padding-left, padding-right and padding-top. As with some other aggregate edge properties, up to four parameter values can be supplied (and if more than one is supplied then the properties are applied to individual borders as described here).

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements included in the *margin* property are:

Value	Description
length	Any specified padding size as a CSS length
%	Padding size specified as percentage of width of container

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.padding="20px 30px"`
Inherited: No
Animatable**:** See *individual properties*

padding-bottom

[CSSPropertyPaddingBottom]

The CSS (CSS1) `padding-bottom` property sets the width of the bottom padding (space) of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
%	As percentage of width of containing element

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.paddingBottom="4px"`
Inherited: No
Animatable**:** Yes

padding-left

[CSSPropertyPaddingLeft]

The CSS (CSS1) `padding-left` property sets the width of the left padding (space) of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
%	As percentage of width of containing element

Default Value: 0
JavaScript syntax: e.g. *object*`.style.paddingLeft="4px"`
Inherited: No
Animatable**:** Yes

padding-right

[CSSPropertyPaddingRight]

The CSS (CSS1) `padding-right` property sets the width of the right padding (space) of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
%	As percentage of width of containing element

Default Value: 0
JavaScript syntax: e.g. *object*`.style.paddingRight="4px"`
Inherited: No
Animatable**:** Yes

padding-top

[CSSPropertyPaddingTop]

The CSS (CSS1) `padding-top` property sets the width of the top padding (space) of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Any specified thickness as a CSS length
%	As percentage of width of containing element

Default Value: 0
JavaScript syntax: e.g. *object*`.style.paddingTop="4px"`
Inherited: No
Animatable**:** Yes

page-break-after

[CSSPropertyPageBreakAfter]

The CSS (CSS2) `page-break-after` property specifies whether a page break should occur after an element. It cannot be used on an empty element or on absolutely positioned elements.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Page breaking defined automatically
`always`	Page break always inserted after element
`avoid`	Where possible avoids a page break after the element
`left`	Page break is inserted after element in a manner that results in the next page being formatted as a left page
`right`	Page break is inserted after element in a manner that results in the next page being formatted as a right page

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.pageBreakAfter="always"`
Inherited: No
Animatable**:** No

page-break-before

[CSSPropertyPageBreakBefore]

The CSS (CSS2) `page-break-before` property specifies whether a page break should occur before an element. It cannot be used on an empty element or on absolutely positioned elements.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Page breaking defined automatically
`always`	Page break always inserted before element
`avoid`	Where possible avoids a page break before the element
`left`	Page break is inserted before element in a manner that results in the next page being formatted as a left page
`right`	Page break is inserted before element in a manner that results in the next page being formatted as a right page

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.pageBreakBefore="always"`
Inherited: No
Animatable**:** No

page-break-inside

[CSSPropertyPageBreakInside]

The CSS (CSS2) `page-break-inside` property specifies whether a page break is allowed inside an element. It cannot be used on absolutely positioned elements.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Page breaking defined automatically
`avoid`	Where possible avoids a page break inside the element

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.pageBreakInside="avoid"`
Inherited: No
Animatable**:** No

perspective

[CSSPropertyPerspective]

The CSS (CSS3) `perspective` property indicates how far a 3D element is notionally placed behind the screen. The property applies to the child elements not the original element itself to which this property is attached.

Valid property values (other than inherit and initial) are:

Value	Description
`length`	Page breaking defined automatically
`none`	(default value). Same as 0, i.e. no perspective set

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.perspective="30px"`
Inherited: No
Animatable**:** Yes

perspective-origin

[CSSPropertyPerspectiveOrigin]

The CSS (CSS3) `perspective-origin` property indicates where a 3D element is notionally placed, based on the *x*-axis and *y*-axis. The property applies to the child elements not the original element itself to which this property is attached. It needs to be used in conjunction with the perspective property and only affects 3D transformed elements.

Valid property values (other than inherit and initial) are:

Value	Description
x-axis y-axis	*x-axis* defines where the view is placed on the *x-axis*, *y-axis* where it is placed on the *y-axis*. Possible values for *x-axis* are: - `left`

	- `center` - `right` - *length* (a CSS length) - *%* (of element size) Possible values for *y-axis* are: - `top` - `center` - `bottom` - *length* (a CSS length) - *%* (of element size)

Default Value: `50% 50%`
JavaScript syntax: e.g. *object*`.style.perspectiveOrigin="20px 40px"`
Inherited: No
Animatable**:** Yes

position

[CSSPropertyPosition]

The CSS (CSS2) `position` property indicates how an element should be positioned.

Valid property values (other than inherit and initial) are:

Value	Description
`absolute`	The element is positioned relative to its first positioned (i.e. not static) ancestor
`fixed`	The element is positioned in a fixed position relative to the browser window
`relative`	The element is positioned relative to its normal position
`static`	(default value). The element is positioned (relative to others) in the order in which it appears in the document flow

Default Value: `static`
JavaScript syntax: e.g. *object*`.style.position="fixed"`
Inherited: No
Animatable**:** No

quotes

[CSSPropertyQuotes]

The CSS (CSS2) `quotes` property indicates how quotation marks should be rendered in the text of an element.

Valid property values (other than inherit and initial) are:

Value	Description
`none`	Open-quote and close-quote elements within an element (i.e. the

	<q> and </q> of a <q> element) will not produce any quotation marks
string1 string2 string3 string4	*String1* and *string2* define the first level of quotation embedding (opening and closing respectively), *string3* and *string4* (if present) the next level of embedding etc.

Default Value: N/A
JavaScript syntax: e.g. *object*`.style.quotes="'\u00AB' '\u00BB'"`
Inherited: No
Animatable**:** No

Common quotation mark characters include:

Representation	Character code used in Javascript syntax	Name
"	`\u0022`	Double quotation mark
'	`\u0027`	Single quotation mark
‘	`\u2018`	Left single high quotation mark
’	`\u2019`	Right single high quotation mark
“	`\u201C`	Left double high quotation mark
”	`\u201D`	Right double high quotation mark
‹	`\u2039`	Left single angle quotation mark
›	`\u203A`	Right single angle quotation mark
«	`\u00AB`	Left double angle quotation mark
»	`\u00BB`	Right double angle quotation mark
„	`\u201E`	Right double low quotation mark

resize

[CSSPropertyResize]

The CSS (CSS3) `resize` property indicates whether an element can be resized by the user. Some major browsers do not support this property.

Valid property values (other than inherit and initial) are:

Value	**Description**
`both`	User can resize height and width of element
`horizontal`	User can resize width of element
`none`	(default value). User cannot resize element
`vertical`	User can resize height of element

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.resize="horizontal"`
Inherited: No
Animatable**:** No

right

[CSSPropertyRight]

The CSS (CSS2) `right` property sets, for absolutely positioned elements, the right edge of an element relative to the corresponding edge of its nearest positioned ancestor. If such an element has no positioned ancestors then it uses the document body and moves along with page scrolling. A 'positioned' element is one whose position is anything other than static.

Valid property values (other than inherit and initial) are:

Value	Description
length	Sets edge position as a CSS length. Can be negative
%	Sets edge position in % of containing element. Can be negative
`auto`	(default value). Browser calculates edge position

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.right="10px"`
Inherited: No
Animatable**:** Yes

tab-size

[CSSPropertyTabSize]

The CSS (CSS3) `tab-size` property indicates size (length) of space used for the tab character.

Valid property values (other than inherit and initial) are:

Value	Description
number	Number of space characters displayed for each tab character
length	Length of tab character (not supported by major browsers)
`none`	User cannot resize element
`vertical`	User can resize height of element

Default Value: `8`
JavaScript syntax: e.g. *object*`.style.tabSize="12"`
Inherited: No
Animatable**:** No

table-layout

[CSSPropertyTableLayout]

The CSS (CSS2) `table-layout` property indicates the algorithm used to define the table layout.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Layout set automatically, with column width set by widest unbreakable content. This can render more slowly as it means all content needs to be read before the layout can be determined)
`fixed`	Layout set only by reference to table's width and width of columns,

	i.e. not by what is in each cell. This can render faster as the browser can begin to display the table as soon as the first row has been received

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.tableLayout="fixed"`
Inherited: No
Animatable**:** No

text-align

[CSSPropertyTextAlign]

The CSS (CSS1) `text-align` property indicates how text in an element should be aligned.

Valid property values (other than inherit and initial) are:

Value	Description
`center`	Centres the text
`justify`	Justifies text, i.e. stretches lines to encompass whole width. The precise way justification then works is set by the text-justify property.
`left`	Aligns text to left
`right`	Aligns text to right

Default Value: `left` if direction is `ltr` (left-to-right), `right` if direction is `rtl` (right-to-left)
JavaScript syntax: e.g. *object*`.style.textAlign="justify"`
Inherited: Yes
Animatable**:** No

text-align-last

[CSSPropertyTextAlignLast]

The CSS (CSS3) `text-align-last` property indicates how the last line of text in an element should be aligned.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Last line is aligned left
`center`	Last line of text is centred
`end`	Last line is aligned to end of line (right if direction is left-to-right, left if direction is right-to-left)
`justify`	Justifies last line of text, i.e. stretches lines to encompass whole width (rarely how text is formatted in practice)
`left`	Aligns last line of text to left
`start`	Last line is aligned to start of line (left if direction is left-to-right, right if direction is right-to-left)

`right`	Aligns last line of text to right

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.textAlignLast="right"`
Inherited: Yes
Animatable**:** No

text-decoration

[CSSPropertyTextDecoration]

The CSS (CSS1) `text-decoration` property indicates the 'decoration' (e.g. underlining) added to text. Note in CSS3 the *text-decoration* property is supposed to be a shorthand property covering text-decoration-line, text-decoration-color and text-decoration-style but at the time of writing this interpretation was not supported by major browsers.

Valid property values (other than inherit and initial) are:

Value	Description
`line-through`	Last line is aligned left
`none`	(default value). Normal text, without decoration
`overline`	Add line above text
`underline`	Add line below text

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.textDecoration="line-through"`
Inherited: No
Animatable**:** No

text-decoration-color

[CSSPropertyTextDecorationColor]

The CSS (CSS3) `text-decoration-color` property indicates the colour of the text decoration added to a given piece of text. For this property to have an effect, a visible text-decoration needs to have been applied to the text.

Valid property values (other than inherit and initial) are:

Value	Description
color	A CSS colour

Default Value: *Colour of text*
JavaScript syntax: e.g. *object*`.style.textDecorationColor="red"`
Inherited: No
Animatable**:** Yes

text-decoration-line

[CSSPropertyTextDecorationLine]

The CSS (CSS3) `text-decoration-line` property indicates the type of line of the text decoration added to a given piece of text. For this property to have an effect, a visible text-decoration needs to have been applied to the text. Multiple values can be combined (e.g. underline and overline)

Valid property values (other than inherit and initial) are:

Value	Description
`line-through`	Last line is aligned left
`none`	(default value). Normal text, without decoration
`overline`	Add line above text
`underline`	Add line below text

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.textDecorationLine="line-through"`
Inherited: No
Animatable**:** No

text-decoration-style

[CSSPropertyTextDecorationStyle]

The CSS (CSS3) `text-decoration-style` property indicates the style of any line in any visible text decoration added to a given piece of text. For this property to have an effect, a visible text-decoration needs to have been applied to the text.

Valid property values (other than inherit and initial) are:

Value	Description
`dashed`	Text decoration displays as a dashed line
`dotted`	Text decoration displays as a dotted line
`double`	Text decoration displays as a double line
`solid`	(default value). Text decoration displays as a single line
`wavy`	Text decoration displays as a wavy line

Default Value: `solid`
JavaScript syntax: e.g. *object*`.style.textDecorationStyle="dotted"`
Inherited: No
Animatable**:** No

text-indent

[CSSPropertyTextIndent]

The CSS (CSS1) `text-indent` property determines the indentation applied to the first line of text in an element. Negative values are allowed (making it possible in effect to indent all but the first line, if the element is sized appropriately).

Valid property values (other than inherit and initial) are:

Value	Description
length	A CSS length
%	Indentation defined in terms of % of width of the parent element

Default Value: `0`
JavaScript syntax: e.g. *object*`.style.textIndent="20px"`
Inherited: Yes
Animatable**:** Yes

text-justify

[CSSPropertyTextJustify]

The CSS (CSS3) `text-justify` property indicates how text is justified when the text-align property has been set to `justify`.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Browser determines justification
`distribute`	Primarily changes spacing at word separators and at grapheme boundaries in scripts other than connected and cursive scripts
`inter-cluster`	Primarily changes spacing at word separators and at grapheme boundaries in cursive scripts
`inter-ideograph`	Primarily changes spacing at word separators and at grapheme boundaries in connected scripts (i.e. ones that use no word spaces)
`inter-word`	Primarily changes spacing at word separators
`kashida`	Primarily stretches Arabic and related scripts through use of kashida or other calligraphic elongation

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.textJustify="distribute"`
Inherited: Yes
Animatable**:** No

text-overflow

[CSSPropertyTextOverflow]

The CSS (CSS3) `text-overflow` property indicates how text that has overflowed is rendered by the browser.

Valid property values (other than inherit and initial) are:

Value	Description
`clip`	(default value). Text is clipped without further action
`ellipsis`	Clipped text is rendered by an ellipsis, i.e. "…"
string	Clipped text is rendered by the given string

Default Value: `clip`
JavaScript syntax: e.g. *object*`.style.textOverflow="ellipsis"`
Inherited: No
Animatable**:** No

text-shadow

[CSSPropertyTextShadow]

The CSS (CSS3) `text-shadow` property indicates what shadow should be added to text. To add more than one shadow, the property should be set to a comma-separated list of shadows.

Valid property values (other than inherit and initial) are:

Value	Description
`none`	(default value). No shadow
h-shadow v-shadow blur-radius color	2 to 4 parameters: - *h-shadow* (required): position of horizontal shadow (can be negative) - *v-shadow* (required): position of vertical shadow (can be negative) - *blur-radius* (optional): how fuzzy (default is zero) - *color* (optional): CSS colour of shadow (default is colour of text)

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.textShadow="2px 10px 5px blue"`
Inherited: Yes
Animatable**:** Yes

text-transform

[CSSPropertyTextTransform]

The CSS (CSS1) `text-transform` property specifies the capitalisation the browser should use for text.

Valid property values (other than inherit and initial) are:

Value	Description
`capitalize`	First character of each word is capitalised (i.e. transformed to uppercase)
`lowercase`	All characters transformed to lowercase
`none`	(default value). No capitalisation
`uppercase`	All characters transformed to uppercase

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.textTransform="capitalize"`
Inherited: Yes
Animatable**:** No

top

[CSSPropertyTop]

The CSS (CSS2) `top` property sets, for absolutely positioned elements, the top edge of an element relative to the corresponding edge of its nearest positioned ancestor. If such an element has no positioned ancestors then it uses the document body and moves along with the page scrolling. A 'positioned' element is one whose position is anything other than static.

Valid property values (other than inherit and initial) are:

Value	Description
length	Sets edge position as a CSS length. Can be negative
%	Sets edge position in % of containing element. Can be negative
`auto`	(default value). Browser calculates edge position

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.top="10px"`
Inherited: No
Animatable**:** Yes

transform

[CSSPropertyTransform]

The CSS (CSS3) `transform` property applies a 2D or a 3D transformation to an element, e.g. rotate, scale, skew transform or translate (move) an element.

Valid property values (other than inherit and initial) are:

Value	Description
`matrix(`*n1*, *n2*, *n3*, *n4*, *n5*, *n6*`)`	2D transformation characterised by 6 values, see below
`matrix3d(`*n1*, *n2*, *n3*, *n4*, *n5*, *n6*, *n7*, *n8*, *n9*, *n10*, *n11*, *n12*, *n13*, *n14*, *n15*, *n16*`)`	3D transformation characterised by 16 values, see below
`none`	(default value). No transformation applied
`perspective(`*n*`)`	A perspective view for a 3D transformed element
`rotate(`*angle*`)`	2D rotation (around origin), *angle* being a CSS angle
`rotate3d(`*x*, *y*, *z*, *angle*`)`	3D rotation (around line through origin), *x*, *y* and *z* being CSS lengths and *angle* being a CSS angle
`rotateX(`*angle*`)`	3D rotation (around *x*-axis), *angle* being a CSS angle
`rotateY(`*angle*`)`	3D rotation (around *y*-axis), *angle* being a CSS angle
`rotateZ(`*angle*`)`	3D rotation (around *z*-axis), *angle* being a CSS angle
`scale(`*x*, *y*`)`	2D scaling transformation, applied to *x* and *y*-axes, *x* and *y* being numbers
`scale3d(`*x*, *y*, *z*`)`	3D scaling transformation, applied to *x*, *y* and *z*-axes, *x*, *y* and *z* being numbers
`scaleX(`*x*`)`	Scaling transformation (stretching / squashing) applied to *x*-axis, *x*

	being numbers
`scaleY(y)`	Scaling transformation (stretching / squashing) applied to *y*-axis, *y* being numbers
`scaleZ(z)`	Scaling transformation (stretching / squashing) applied to *z*-axis, *z* being numbers
`skew(x-angle, y-angle)`	2D skew transformation along *x* and *y*-axes, *x-angle* and *y-angle* being CSS angles
`skewX(angle)`	Skew transformation along *x*-axis, *angle* being a CSS angle
`skewY(angle)`	Skew transformation along *y*-axis, *angle* being a CSS angle
`translate(x, y)`	2D translation, applied to *x* and *y*-axes, *x* and *y* being CSS lengths
`translate3d(x, y, z)`	3D translation, applied to *x*, *y* and *z*-axes, *x*, *y* and *z* being CSS lengths
`translateX(x)`	Translation (movement) applied to *x*-axis, *x* being a CSS length
`translateY(y)`	Translation (movement) applied to *y*-axis, *y* being a CSS length
`translateZ(z)`	Translation (movement) applied to *z*-axis, *z* being a CSS length

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.transform="translateX(30px)"`
Inherited: No
Animatable**:** Yes

2D transformations

There are 6 types of 2D transformations: `translate()`, `rotate()`, `scale()`, `skewX()`, `skewY()` and `matrix()`. These have the following characteristics:

- `translate()`: moves an element from its current position along the *x* and *y*-axes, but doesn't otherwise change its shape or size
- `rotate()`: rotates an element clockwise (positive) or counter-clockwise (negative), e.g. rotate(20deg)
- `scale()`: increases or decreases the size of an element (parameter is a ratio relative to the original size), first parameter is width (i.e. *x*-axis) scaling, second is height (i.e. *y*-axis) scaling
- `skewX()`: introduces a skew along the *x*-axis, by a given angle (positive or negative)
- `skewY()`: introduces a skew along the *y*-axis, by a given angle (positive or negative)
- `skew(x-angle, y-angle)`: combines skewX(*x-angle*) and skewY(*y-angle*)
- `matrix()`: combines all 2D transform methods into a single method, involving the following parameters: `matrix(scale-x, skew-y, skew-x, scale-y, translate-x, translate-y)`

The origin of the transformation is defined by the transform-origin property.

3D transformations

Not currently covered in this page.

transform-origin

[CSSPropertyTransformOrigin]

The CSS (CSS3) `transform-origin` property defines the origin used by the transform property.

Valid property values (other than inherit and initial) are:

Value	Description
x-axis y-axis z-axis	Origin defined by three values, where: *x-axis* can be: - *length* (a CSS length) - *%* (of overall element size) - `left` (origin *x*-coordinate at left end of element) - `centre` (origin *x*-coordinate at centre of element) - `right` (origin *x*-coordinate at right end of element) *y-axis* can be: - *length* (a CSS length) - *%* (of overall element size) - `bottom` (origin *y*-coordinate at bottom end of element) - `centre` (origin *y*-coordinate at centre of element) - `top` (origin *y*-coordinate at right end of element) *z-axis* can be: - *length* (a CSS length)

Default Value: `50% 50% 0`
JavaScript syntax: e.g. *object*`.style.transformOrigin="0 0"`
Inherited: No
Animatable**:** Yes

transform-style

[CSSPropertyTransformStyle]

The CSS (CSS3) `transform-style` property indicates how nested elements are to be rendered for 3D purposes when using the transform property.

Valid property values (other than inherit and initial) are:

Value	Description
`flat`	(default value). Child elements do not preserve their 3D position
`preserve-3d`	Child elements preserve their own 3D position

Default Value: `flat`
JavaScript syntax: e.g. *object*`.style.transformStyle="preserve-3d"`
Inherited: No
Animatable**:** No

transition

[CSSPropertyTransition]

The CSS (CSS3) `transition` property is a shorthand property combining the 4 transition sub-properties.

Valid property values (other than inherit and initial) are defined by the elements of the shorthand. Shorthand elements (in the order in which they appear):

- transition-property
- transition-duration
- transition-timing-function
- transition-delay

Default Value: See *individual properties*
JavaScript syntax: e.g. *object*`.style.transition="all 5s"`
Inherited: No
Animatable**:** No

transition-delay

[CSSPropertyTransitionDelay]

The CSS (CSS3) `transition-delay` property indicates when a transition will start.

Valid property values (other than inherit and initial) are:

Value	Description
time	The CSS time to wait before transition effect starts

Default Value: `0s`
JavaScript syntax: e.g. *object*`.style.transitionDelay="5s"`
Inherited: No
Animatable**:** No

transition-duration

[CSSPropertyTransitionDuration]

The CSS (CSS3) `transition-duration` property indicates how long a transition will take to complete.

Valid property values (other than inherit and initial) are:

Value	Description
time	The CSS time a transition will take to complete once started

Default Value: `0s`
JavaScript syntax: e.g. *object*`.style.transitionDuration="4s"`
Inherited: No
Animatable**:** No

transition-property

[CSSPropertyTransitionProperty]

The CSS (CSS3) `transition-property` property identifies the properties that change as part of a transition effect. You should always specify the transition-duration property as well, as otherwise it defaults to zero.

Valid property values (other than inherit and initial) are:

Value	Description
property	Comma separated list of CSS properties to which transition is applied
`all`	(default value). Transition effect is applied to all CSS properties
`none`	Transition effect is applied to no properties

Default Value: `all`
JavaScript syntax: e.g. *object*`.style.transitionProperty="width"`
Inherited: No
Animatable**:** No

transition-timing-function

[CSSPropertyTransitionTimingFunction]

The CSS (CSS3) `transition-timing-function` property identifies the speed curve used for a transition effect.

Valid property values (other than inherit and initial) are:

Value	Description
`cubic-bezier(`*x1*, *x2*, *x3*, *x4*`)`	As defined by cubic Bezier function with parameters *x1*, *x2*, *x3*, *x4* (possible values of each are numerical values from `0` to `1`)
`ease`	(default value). Slow start, then fast, before ending slowly, equivalent to cubic-bezier(0.25,0.1,0.25,1)
`ease-in`	Slow start, equivalent to cubic-bezier(0.42,0,1,1)
`ease-out`	Slow end, equivalent to cubic-bezier(0,0,0.58,1)
`ease-in-out`	Slow start and end, equivalent to cubic-bezier(0.42,0,0.58,1)
`step-start`	Equivalent to `steps(1, start)`
`step-end`	Equivalent to `steps(1, end)`
`steps(int,start|end)`	A stepping function with two parameters. The first parameter specifies the number of intervals in the function (and must be a positive integer, i.e. greater than zero). The second (optional) parameter specifies when the change of values occurs and is either `start` or `end` (if omitted is given the value `end`)

Default Value: `ease`
JavaScript syntax: e.g. *object*`.style.transitionTimingFunction="linear"`
Inherited: No
Animatable**:** No

unicode-bidi

[CSSPropertyUnicodeBidi]

The CSS (CSS2) `unicode-bidi` property indicates whether text direction should be overridden to support multiple languages in the same document. It is used in conjunction with the direction property.

Valid property values (other than inherit and initial) are:

Value	Description
`bidi-override`	Creates an additional level of embedding and reorders depending on direction property
`embed`	Creates an additional level of embedding
`normal`	(default value). No additional level of embedding

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.unicodeBidi="bidi-override"`
Inherited: Yes
Animatable**:** No

user-select

[CSSPropertyUserSelect]

The CSS (CSS3) `user-select` property indicates whether text of an element can be selected. If you (double) click on some text it will typically be selected, and this property stops this happening.

Valid property values (other than inherit and initial) are:

Value	Description
`auto`	(default value). Text can be selected by user (if allowed by browser)
`none`	Text can't be selected by user
`text`	Text can be selected by user

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.userSelect="none"`
Inherited: No
Animatable**:** No

vertical-align

[CSSPropertyVerticalAlign]

The CSS (CSS1) `vertical-align` property indicates the vertical alignment of an element.

Valid property values (other than inherit and initial) are:

Value	Description
length	Raises (positive) or lowers (negative) element by a CSS length
%	Raises of lowers element by percentage of line-height property
`baseline`	(default value). Baseline of element aligned with baseline of parent
`bottom`	Bottom of element aligned with bottom of lowest element on line

middle	Element is placed vertically in middle of parent
sub	Aligns element as if it was a subscript
super	Aligns element as if it was a superscript
text-bottom	Bottom of element aligned with bottom of parent text
text-top	Top of element aligned with top of parent text
top	Top of element aligned with top of highest element on line

Default Value: baseline
JavaScript syntax: e.g. *object*.style.verticalAlign="sub"
Inherited: No
Animatable: Yes

visibility

[CSSPropertyVisibility]

The CSS (CSS2) visibility property indicates whether an element is visible or not. Note: Invisible (hidden) elements still take up some space on a page; if you want to avoid this then set the display property to none.

Valid property values (other than inherit and initial) are:

Value	Description
collapse	Only applies to table elements. Row or column is removed, but table layout is otherwise left unaltered. For non-table elements is equivalent to "hidden"
hidden	Element is hidden (but still takes up space)
visible	(default value). Element is visible

Default Value: visible
JavaScript syntax: e.g. *object*.style.visibility="hidden"
Inherited: No
Animatable: Yes

white-space

[CSSPropertyWhiteSpace]

The CSS (CSS1) white-space property indicates how white-space inside an element should be handled.

Valid property values (other than inherit and initial) are:

Value	Description
normal	(default value). Sequences of whitespace collapsed into a single whitespace and text will wrap when necessary
nowrap	Sequences of whitespace collapsed into a single whitespace but text will not wrap until a tag occurs
pre	Whitespace is preserved by browser and text will only wrap on line breaks (i.e. akin to <pre> tag in HTML)

`pre-line`	Sequences of whitespace collapsed into a single whitespace and text will wrap when necessary and on line breaks
`pre-wrap`	Whitespace is preserved by browser and text will wrap when necessary and on line breaks

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.whiteSpace="hidden"`
Inherited: Yes
Animatable**:** No

widows

[CSSPropertyWidows]

The CSS (CSS3) `widows` property indicates the minimum number of lines of a paragraph that can fall to a new page. It works primarily with paged media, in which content is split into pages.

Valid property values (other than inherit and initial) are:

Value	Description
number	Integer

Default Value: `2`
JavaScript syntax: e.g. *object*`.style.widows="3"`
Inherited: No
Animatable**:** Yes

width

[CSSPropertyWidth]

The CSS (CSS1) `width` property indicates the width of an element (excluding padding, borders and margins). It is overridden by the min-width or max-width properties, if either of them are present.

Valid property values (other than inherit and initial) are:

Value	Description
length	Width as a CSS length
%	Width of element defined as a percentage of width of its containing block
`auto`	(default value). Browser determines width

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.width="300px"`
Inherited: No
Animatable**:** Yes

word-break

[CSSPropertyWordBreak]

The CSS (CSS3) `word-break` property indicates the way in which words can be broken at line ends for scripts that are not Chinese, Japanese or Korean ("CJK").

Valid property values (other than inherit and initial) are:

Value	Description
`break-all`	Breaks can occur between any two letters
`keep-all`	Breaks are prohibited between pairs of letters
`normal`	(default value). Words break at line ends according to usual rules

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.wordBreak="keep-all"`
Inherited: Yes
Animatable**:** No

word-spacing

[CSSPropertyWordSpacing]

The CSS (CSS1) `word-spacing` property indicates the amount of whitespace between words.

Valid property values (other than inherit and initial) are:

Value	Description
length	Additional space between words as a CSS length, can be negative
`normal`	(default value). Normal space between words

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.wordSpacing="10px"`
Inherited: Yes
Animatable**:** Yes

word-wrap

[CSSPropertyWordWrap]

The CSS (CSS3) `word-wrap` property allows long words to be broken at line ends and to wrap onto the next line.

Valid property values (other than inherit and initial) are:

Value	Description
`break-word`	Otherwise unbreakable words can be broken
`normal`	(default value). Words can only be broken at allowed break points

Default Value: `normal`
JavaScript syntax: e.g. *object*`.style.wordWrap="break-word"`
Inherited: Yes
Animatable**:** No

z-index

[CSSPropertyZIndex]

The CSS (CSS1) `z-index` property specifies the stack order of an element, i.e. which is "in front of" other elements, and hence which is visible if several would otherwise appear in the same place. Elements with higher stack order (z-index value) are shown in preference to ones with lower stack order. It only works on positioned elements, i.e. with `position:absolute`, `position:relative` or `position:fixed`.

Valid property values (other than inherit and initial) are:

Value	Description
number	Stack order set to a specified value (which can be negative)
`auto`	(default value). Stack order of element set to the same as its parent's stack order

Default Value: `auto`
JavaScript syntax: e.g. *object*`.style.zIndex="-1"`
Inherited: No
Animatable**:** Yes

Appendix D: CSS Shorthand Properties

[CSSShorthandProperty]

Some of the CSS properties are shorthand properties that combine several related properties, e.g.:

Shorthand rule:	E.g. `div {border:` *border-width border-style border-color*`;}`

Is equivalent to	`div {`
	`  border-width:` *border-width*;
	`  border-style:` *border-style*;
	`  border-color:` *border-color*;
	`}`

Further comments	Shorthand properties can typically also take the value initial or inherit

Shorthand properties are:

- animation
- background
- border
- border-bottom
- border-image
- border-left
- border-right
- border-top
- column-rule
- columns
- flex
- flex-flow
- font
- grid
- list-style
- margin
- outline
- padding
- transition

Appendix E: CSS Animatable Properties

[CSSAnimatableProperties]

Some CSS properties are *animatable* under CSS3. This means that they can be used in animations and transitions. These properties can be changed gradually from one value to another. Properties that are animatable are:

- background, background-color, background-position, background-size
- border, border-bottom, border-bottom-color, border-bottom-left-radius, border-bottom-right-radius, border-bottom-width, border-color, border-left, border-left-color, border-left-width, border-right, border-right-color, border-right-width, border-spacing, border-top, border-top-color, border-top-left-radius, border-top-right-radius, border-top-width
- bottom
- box-shadow
- clip
- color
- column-count, column-gap, column-rule, column-rule-color, column-rule-width, column-width
- columns
- filter
- flex, flex-basis, flex-grow, flex-shrink
- font, font-size, font-size-adjust, font-stretch, font-weight
- height
- left
- letter-spacing
- line-height
- margin, margin-bottom, margin-left, margin-right, margin-top
- max-height, max-width
- min-height, min-width
- opacity
- order
- outline, outline-color, outline-offset, outline-width
- padding, padding-bottom, padding-left, padding-right, padding-top
- perspective, perspective-origin
- right
- text-decoration-color, text-indent, text-shadow
- top
- transform, transform-origin
- vertical-align
- visibility
- width
- word-spacing
- z-index

Appendix F: CSS Keywords (inherit and initial)

inherit

[CSSKeywordInherit]

The CSS (CSS3) `inherit` keyword is used to specify that an element should inherit its value from its parent element.

For example, the following means the color property for elements should be `red`, except for those which have `class = "colorinherited"`, which would inherit theirs from their parent element

Example rule:

```
span {color: red; }
span.colorinherited {color: inherit;}
```

JavaScript syntax: e.g. *object*`.style.`*property*`="inherit"`

initial

[CSSKeywordInitial]

The CSS (CSS3) `initial` keyword is used to set a CSS property to its default value. It can be used for any CSS property and on any HTML element, e.g.:

Example rule:

```
div {color: initial; }
```

JavaScript syntax: e.g. *object*`.style.`*property*`="initial"`

Appendix G: CSS Pseudo-Properties (content, counter-increment and counter-reset)

content

[CSSPseudoPropertyContent]

The CSS (CSS2) `content` (pseudo-)property is used with the `:before` and `:after` pseudo-elements to insert generated content.

For example, a selector taking the following form will add the relevant web address to the link

```
a:after { content: " (i.e. " attr(href) ")"; }
```

Property values (other than inherit and initial) that can be included in the pseudo-property include:

Value	Description
string	Some specified text
`attr(`*attribute*`)`	A specified attribute
`close-quote`	A closing quote
`counter(`*id*`)`	A counter with id defined by *id* (see counter-increment and counter-reset pseud-properties)
`no-close-quote`	Removes closing quote of content, if present
`no-open-quote`	Removes opening quote of content, if present
`none`	Nothing
`normal`	(Default). Sets content, if specified to normal, i.e. none
`open-quote`	An opening quote
`url(`*URL*`)`	URL specifying a media (image, sound, video etc.) to be included in the pseudo-property

Default Value: `normal`
JavaScript syntax: N/A. You can't give an element a pseudo-class (but you can manipulate the document in ways that achieve a similar effect)
Inherited: No
Animatable**:** No

counter-increment

[CSSPseudoPropertyCounterIncrement]

The CSS (CSS2) `counter-increment` (pseudo-)property increments one or more CSS counter values and is usually used in conjunction with the counter-reset and content pseudo-properties.

For example, a selector taking the following form will increment the counter named `ctr` by 2 each time the relevant selector is selected (here each time the page load comes across an <h2> element)

```
h2 { counter-increment: ctr 2; }
```

Property values (other than inherit and initial) that can be included in the pseudo-property include:

Value	Description
id value	The *id* of the counter to be incremented and the *value* that the counter is to be incremented by (can be negative or zero, default is 1)
`none`	(default). No counters incremented

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.counterIncrement="ctr"`
Inherited: No
Animatable**:** No

counter-reset

[CSSPseudoPropertyCounterReset]

The CSS (CSS2) `counter-reset` (pseudo-)property creates or resets one or more CSS counters and is usually used in conjunction with the counter-increment and content pseudo-properties.

For example, a selector taking the following form will reset the counter named `ctr` to 1 each time the relevant selector is selected (here each time the page load comes across an <h1> element)

```
h1 { counter-reset: ctr; }
```

Property values (other than inherit and initial) that can be included in the pseudo-property include:

Value	Description
id value	The *id* of the counter to be reset (created) and the *value* that the counter is to be reset to on each occurrence of the selector (default value is 0)
`none`	(default). No counters reset

Default Value: `none`
JavaScript syntax: e.g. *object*`.style.counterReset="ctr"`
Inherited: No
Animatable**:** No

Appendix H: CSS Rules (@font-face, @keyframes, @media)

@font-face

[CSSRuleFontFace]

The CSS (CSS3) `@font-face` rule allows designers to apply their own font. Syntax is as follows:

```
@font-face {
    font-properties
}
```

The *font-properties* are:

Descriptor	Description / Values it can take
`font-family`	Required. Name of font
`src`	Required. A valid URL from which the font is downloaded
`font-stretch`	Optional. Akin to the CSS font-stretch property. Indicates how the font should be stretched. Acceptable values are (default is `normal`): - `condensed` - `expanded` - `extra-condensed` - `extra-expanded` - `normal` - `semi-condensed` - `semi-expanded` - `ultra-condensed` - `ultra-expanded`
`font-style`	Optional. Akin to the CSS font-style property. Indicates how the font to be styled. Acceptable values are (default is `normal`): - `italic` - `normal` - `oblique`
`font-weight`	Optional. Akin to the CSS font-weight property. Indicates boldness of font. Acceptable values are (default is `normal`): `normal`, `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900`
`unicode-range`	Optional. Indicates range of Unicode characters that the font supports (default is `U+0-10FFFF`):

@keyframes

[CSSRuleKeyframes]

The CSS (CSS3) `@keyframes` rule is the way in which designers specify animations that use CSS animation properties. Syntax is as follows:

```
@keyframes name {
    keyframes-selector {css-styles;}
}
```

The components are:

Descriptor	Description / Values it can take
name	Required. Name of animation
keyframes-selector	At least one required, but for an animation to apply you need to include more than one. Percentage of the animation duration. Acceptable values are: - `0` to `100%` - `from` (is the same as 0%) - `to` (is the same as 100%
css-styles	Required. One or more CSS style properties.

@media

[CSSRuleMedia]

The CSS (CSS2 / CSS3) `@media` rule is used to apply different styles for different devices and/or media types. Syntax is as follows:

```
@media not|only media type and|not|only (media feature) {
  CSS-Code
}
```

Style sheets can also be applied to different media using e.g.

```
<link rel="stylesheet" media="xxx" href="stylesheet.css" >
```

Recognised (non-depreciated) *media types* include:

Value	Description
`all`	All media types
`print`	Printers
`screen`	Computer screens, tablets, smartphones etc.
`speech`	Screen readers that read out loud the page contents

Depreciated media types include: `aural`, `braille`, `embossed`, `handheld`, `projection`, `tty`, `tv`

Recognised media features include:

Value	Description
`any-hover`	Does any available input mechanism allow user to hover over elements?
`any-pointer`	Does any available input mechanism allow user to point, and if so, how accurate is it?
`aspect-ratio`	All media types
`color`	Number of bits per colour component handled by device
`color-index`	Number of colours device displays
`device-aspect-ratio`	Ratio between width and height of device (depreciated)
`device-height`	Height of device (depreciated)

device-width	Width of device (depreciated)
grid	Identifies whether device is a grid or bitmap
height	Display height
hover	Does primary input mechanism allow user to hover over elements?
inverted-colors	Does browser / underlying operating system support inverting of colours
light-level	Current ambient light level
max-aspect-ratio	Maximum ratio between width and height of device
max-color	Maximum number of bits per colour component device can handle
max-color-index	Maximum number of colours device can display
max-device-aspect-ratio	Maximum ratio between width and height of device
max-device-height	Maximum height of device
max-device-width	Maximum width of device
max-height	Maximum display height
max-monochrome	Maximum number of bits per colour on a monochrome device
max-resolution	Maximum resolution of device (using dpi or dpcm)
max-width	Maximum display width
min-aspect-ratio	Minimum ratio between width and height of device
min-color	Minimum number of bits per colour component device can handle
min-color-index	Minimum number of colours device can display
min-device-aspect-ratio	Minimum ratio between width and height of device
min-device-height	Minimum height of device
min-device-width	Minimum width of device
min-height	Minimum display height
min-monochrome	Minimum number of bits per colour on a monochrome device
min-resolution	Minimum resolution of device (using dpi or dpcm)
min-width	Minimum display width
monochrome	Number of bits per colour on a monochrome device
orientation	Whether device is in landscape or portrait mode
overflow-block	How device handles content that overflows along block axis
overflow-inline	How device handles content that overflows along inline axis
pointer	Does primary input mechanism allow user to point, and if so, how accurate is it?
resolution	Resolution of device (using dpi or dpcm)
scan	Scanning process of device
scripting	Is scripting (typically JavaScript) available?
update-frequency	How quickly device can change appearance
width	Display width

Appendix I: CSS Selectors

[CSSSelector]

Commonly CSS is applied to all elements of a specific type. By using selectors, we can, however, apply CSS to a wide range of sub-types, selected in a wide variety of ways. Some selectors (e.g. the `:hover` selector) depend on mouse position or activity. The following are valid selector types:

Selector	Description*	Example
`*`	Selects all elements	`*`
#id	Selects the element with a given id attribute (e.g. `id="yyy"`)	`#yyy`
Element	Selects all elements of type *element*	`p`
element.class	Selects all elements of type *element* with a given *class* attribute (e.g. `class="xxx"`)	`p.xxx`
element1, *element2*	Selects all elements of either type *element1* or type *element2*	`p, a`
element1 element2	Selects all elements of type *element2* that are inside an element of type *element1*	`p a`
element1>*element2*	Selects all elements of type *element2* that have as their parent an element of type *element1*	`p>a`
element1+*element2*	Selects all elements of type *element2* that are immediately after elements of type *element1*	`p+a`
element1~*element2*	Selects all elements of type *element2* that are preceded by an element of type *element1*	`p+a`
element[*attribute*]	Selects all elements of type *element* with a specific attribute*	`p[target]`
element[*attribute* = *value*]	Selects all elements of type *element* that have a specific attribute taking a specific value	`p[target = "_blank"]`
element[*attribute* ~= *word*]	Selects all elements of type *element* that have a specific attribute containing a specific word	`p[title ~= "answer"]`
element[*attribute* |= *word*]	Selects all elements of type *element* that have a specific attribute starting with a specific word. The word needs to be either alone, like `lang=en`, or followed by a hyphen(-), like `lang=en-us`.	`p[title |= "answer"]`
element[*attribute* ^= *value*]	Selects all elements of type *element* that have a specific attribute starting with a specific value	`a[href ^= "https"]`
element[*attribute* $= *value*]	Selects all elements of type *element* that have a specific attribute ending with a specific value	`a[href $= ".pdf"]`
element[*attribute* *= *value*]	Selects all elements of type *element* that have a specific attribute containing a specific sub-string	`a[href *= "Nematrian"]`
element:`active`	Selects whatever element of type *element* is currently active	`a:active`
element:`checked`	Selects all elements of type *element* that are checked	`input:checked`
element:`disabled`	Selects all elements of type *element* that are disabled	`input:disabled`
element:`empty`	Selects all elements of type *element* that are empty	`div:empty`
element:`enabled`	Selects all elements of type *element* that are	`input:enabled`

	enabled	
element:`first-child`	Selects all elements of type *element* that are the first children of their parent	`p:first-child`
element:`first-of-type`	Selects all elements of type *element* that are the first of that type of element within their parent	`p:first-of-type`
element:`focus`	Selects all elements of type *element* that has focus**	`input:focus`
element:`hover`	Selects all elements of type *element* that are currently being hovered over (i.e. where mouse is positioned over it)	`a:hover`
element:`in-range`	Selects all elements of type *element* whose value is within any range specified by the element	`input:in-range`
element:`invalid`	Selects all elements of type *element* with an invalid value	`input:invalid`
element:`lang(`*language*`)`	Selects all elements of type *element* with a `lang` value equal to *language*	`p:lang(it)`
element:`last-child`	Selects all elements of type *element* that are the last children of their parent	`p:last-child`
element:`last-of-type`	Selects all elements of type *element* that are the last of that type of element within their parent	`p:last-of-type`
element:`link`	Selects all elements of type *element* that are unvisited	`a:link`
`:not(`*selector*`)`	Selects all elements that are not the given *selector*	`:not(p)`
element:`nth-child(`*n*`)`	Selects all elements of type *element* that are the *n*'th child of their parent	`a:nth-child(2)`
element:`nth-last-child(`*n*`)`	Selects all elements of type *element* that are the *n*'th last child (i.e. counting backwards from the last one) of their parent	`a:nth-last-child(2)`
element:`nth-last-of-type(`*n*`)`	Selects all elements of type *element* that are the *n*'th last of their type (i.e. counting backwards from last one) of their parent	`a:nth-last-of-type(2)`
element:`nth-of-type(`*n*`)`	Selects all elements of type *element* that are the n'th of their type of their parent	`a:nth-of-type(2)`
element:`only-child`	Selects all elements of type *element* that are the only child of their parent	`a:only-child`
element:`only-of-type`	Selects all elements of type *element* that are the only one of their type of their parent	`a:only-of-type`
element:`optional`	Selects all elements of type *element* that do not have a `required` attribute specified	`input:optional`
element:`out-of-range`	Selects all elements of type *element* whose value is outside any range specified by the element	`input:out-of-range`
element:`read-only`	Selects all elements of type *element* which have the `readonly` attribute specified	`input:read-only`
element:`read-write`	Selects all elements of type *element* which do not have the `readwrite` attribute specified	`input:read-write`
element:`required`	Selects all elements of type *element* that have a `required` attribute specified	`input:required`

`:root`	Selects the document's root element	`:root`
element`:target`	Selects the current active target element. A URL with an # followed by an anchor links to a specific element within a document. The element being linked to is the target element.	`#para1:target`
element`:valid`	Selects all elements of type *element* that have a valid value	`input:valid`
element`:visited`	Selects all visited elements	`a:visited`

* If *element* is left out of the selector then the selector applies to all element types
** The element that has focus is the one that if you hit return will be assumed to be the element into which input has just been placed.

A handful of selectors don't apply to elements but to components of elements:

Selector	**Description***	**Example**
element`::after`	Inserts material immediately after the content of the elements of type *element*	`h1::after`
element`::before`	Inserts material immediately before the content of the elements of type *element*	`h1::before`
element`::first-letter`	Selects first letter of elements of type *element*	`p::first-letter`
element`::first-line`	Selects first line of elements of type *element*	`p::first-line`
element`::selection`	Selects the portion of elements of type *element* that are selected by the user	`::selection`

Appendix J: CSS Units: Times, Lengths, Angles and Colours

CSS Times

[CSSTime]

Some CSS properties relate to periods of time. CSS time units are defined in seconds (s) or milliseconds (ms), e.g. `2s` or `600ms`.

CSS Lengths

[CSSLength]

Often it is important to specify the size of an HTML element. The conventions used when doing this in CSS are set out below.

A CSS length is a number followed by a length unit, such as 20px or 3cm. To be correctly understood, the specification needs to avoid any whitespace between the number and the length unit (e.g. using `20 px` will generally not be recognised as a length by browsers). If the value is zero (`0`) then the unit can be omitted.

Length units can be absolute or absolute.

Absolute lengths

These are fixed in size, and material will should appear exactly that size (unless the user then zooms in or out manually). As screen sizes vary considerably, best practice typically recommends using relative lengths not absolute lengths.

Unit	Description
`cm`	centimetres
`mm`	millimetres
`in`	inches (1in = 2.54cm)
`px`	pixels (1px = 1/96th of 1in, for high resolution screens, but for low resolution screens then 1px is one device pixel, i.e. dot, of the display
`pt`	points (1pt = 1/72 of 1in)
`pc`	picas (1pc = 12pt)

Relative lengths

Relative lengths are specified relative to another length property. These types of lengths tend to scale better across different screens or other rendering mediums.

Unit	Description
`ch`	Relative to width of a “0” (zero) character
`em`	Relative to font-size of element (e.g. 2em means twice the relevant font size)
`ex`	Relative to x-height of current font (this unit is rarely used)
`rem`	Relative to font-size of root element
`vw`	Relative to 1% of the width of the viewport (i.e. the browser window size)
`vh`	Relative to 1% of the height of the viewport (i.e. the browser window size)

vmin	Relative to 1% of the viewport's smaller dimension (not recognised by all browsers)
vmax	Relative to 1% of the viewport's larger dimension (not recognised by all browsers)

CSS Angles

[CSSAngle]

Some CSS property values are defined in terms of angles. It is used for example in rotate or skew parameters used by the transform property.

Angles can be defined in the following units:

Unit	Description
deg	Degrees. One full circle (360°) is 360deg
grad	Gradians. One full circle (360°) is 400deg
rad	Radians. One full circle (360°) is 2π radians, i.e. approximately 6.28318rad
turn	Number of full turns. One full circle (360°) is 1turn. The turn unit is not at the time of writing supported by all major browsers.

Positive angles represent clockwise turns, whilst negative angles represent counter-clockwise turns

Note: as with CSS lengths, no space should be left between the numerical value and unit. Unlike with CSS lengths, you always need to include a unit, i.e. 0 is not in itself a valid angle, even though 0deg = 0grad = 0rad = 0turn.

CSS Colours

[CSSColor]

A common activity within CSS (and any associated HTML markup) is to set the colour of an element. Colours can be specified in several ways:

- Using predefined names
- Using hexadecimal format
- Using RGB or RGBA formats
- Using HSL or HSLA formats

1. Predefined names

The following specific colours are listed in HTML / CSS specifications:

Colour name	Hexadecimal value	Colour
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	

Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGrey	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
DarkOrange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	
DarkSlateGrey	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	

DimGrey	#696969	
DodgerBlue	#1E90FF	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	
Grey	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFF0	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGray	#D3D3D3	
LightGrey	#D3D3D3	
LightGreen	#90EE90	
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateGray	#778899	
LightSlateGrey	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	

Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370DB	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	
OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	
PaleVioletRed	#DB7093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
RebeccaPurple	#663399	
Red	#FF0000	

RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	
Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
SlateGrey	#708090	
Snow	#FFFAFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	

2. Hexadecimal format

All major browsers recognise hexadecimal format ('hex') colour values. These take the form `#RRGGBB` where the RR, GG and BB indicate the red, green and blue components of the colour in two-character hexadecimal format (i.e. ranging from 00 to FF, which in decimal correspond to 0 to 255 respectively. For example, `#FF0000` is red, since the red component is given its maximum value (FF, i.e. 255 in decimal format), whilst the green and blue components are given their minimum values, i.e. do not appear in the end colour. The hexadecimal format codes for each of the prespecified CSS colours are set out above.

3. RGB and RGBA formats

RGB colour values are also recognised by all major browsers. They are specified in the form `rgb`(*r*,*g*,*b*), where *r*, *g* and *b* are the red, green and blue components of the colour as above, but specified either in decimal values (0 to 255) or in percentage values (0% to 100%)

RGBA colour values are recognised by many major browsers. They extend the RGB to include an opacity (i.e. transparency) value. They are specified in the form `rgba`(*r*,*g*,*b*,*a*), where *r*, *g* and *b* are as above, and a (the 'alpha' parameter) can take a value between `0.0` (fully transparent, so invisible) and `1.0` (fully opaque, so will entirely block whatever is 'behind' the element assigned this opacity).

3. HSL and HSLA formats

HSL colour values are recognised by many browsers. HSL stands for hue, saturation and lightness. Such a colour is specified by `hsl`(*h*,*s*,*l*) where *h* refers to the hue, *s* the saturation and *l* the lightness of the colour.

HSL can be thought of as a cylindrical-coordinate representation of a colour. Hue defines the primary colour or mix (in degrees), between 0 and 360, with 0 (or 360) corresponding to red, 120 corresponding to green and 240 corresponding to blue (with intermediate values corresponding to mixtures of these colours). Saturation corresponds to the extent to which the colour diverges from grey, and is expressed in percentage terms, where 0% corresponds to grey and 100% to full colour. Lightness is also expressed in percentage terms, where 0% corresponds to black and 100% to white.

HSLA is like RGBA in extending the colour format to include an opacity value. It is specified by `hsla`(*h*,*s*,*l*,*a*), where *a* is the alpha parameter as above.

Appendix K: Miscellaneous CSS Property Values (Border Styles and Edge Multi-Value Formats)

CSS Border Styles

[CSSBorderStyle]

CSS *border-style* properties (i.e. border-style, border-bottom-style, border-left-style, border-right-style and border-top-style) can take the following values (other than inherit and initial):

Value	Description
`dashed`	Dashed border
`dotted`	Dotted border
`double`	Double border
`groove`	Effect depends on relevant border-color value (i.e. value of border-color, border-bottom-color, border-left-color, border-right-color or border-top-color
`hidden`	Same as "none" except when there are border conflicts
`inset`	Effect depends on relevant border-color value
`none`	(default value). No border
`outset`	Effect depends on relevant border-color value
`ridge`	Effect depends on relevant border-color value
`solid`	Solid border

CSS Edge Multi-Value Formats

[CSSEdgeMultiValueFormat]

Several aggregate CSS edge (border, margin, padding) properties can take multiple values. Depending on the number of values supplied the rule for deciding which value is applied to which edge is given below:

Number supplied	E.g.	Which values are applied to which edge
1	*x1*	*x1* applied to all four edges
2	*x1 x2*	*x1* applied to top and bottom edges *x2* applied to right and left edges
3	*x1 x2 x3*	*x1* applied to top edge *x2* applied to right and left edges *x3* applied to bottom edge
4	*x1 x2 x3 x4*	*x1* applied to top edge *x2* applied to right edge *x3* applied to bottom edge *x4* applied to left edge

Appendix L: Default CSS Styles Applied to HTML Elements

[HTMLCSSDefaults]

The default CSS styles applied to different renderable HTML elements supported by HTML 5 are set out below.

Some element types have no applicable default CSS. These include:

HTML Element(s)	Default CSS applicable to that element
<abbr>, <audio>, <base>, <bdi>, , <button>, <canvas>, <data>, <dialog>, <input>, <keygen>, <main>, <menuitem>, <meta>, <meter>, <noscript>, <optgroup>, <option>, <progress>, <rp>, <ruby>, <select>, <source>, , <textarea>, <time>, <track>, <video>, <wbr>	None

For those that do have a default, occasionally this is browser specific, but usually it follows certain conventions:

HTML Element(s)	More	Default CSS applicable to that element

`a:link`	<a>	`color:` (*is an internal value specific to browser*)`;`
		`text-decoration: underline;`
		`cursor: auto;`

`a:visited`	<a>	`color:` (*is an internal value specific to browser*)`;`
		`text-decoration: underline;`
		`cursor: auto;`

`a:link:active`	<a>	`color:` (*is an internal value specific to browser*)`;`
`a:visited:active`	<a>	`color:` (*is an internal value specific to browser*)`;`
`address`	<address>	`display: block;` `font-style: italic;`
`area`	<area>	`display: none;`
`article`	<article>	`display: block;`
`aside`	<aside>	`display: block;`
`b`	<b>	`font-weight: bold;`
`bdo`	<bdo>	`unicode-bidi: bidi-override;`
`blockquote`	<blockquote>	`display: block;` `margin: 1em 40px 1em 40px;`
`body`	<body>	`display: block;` `margin: 8px;`
`body:focus`	<body>	`outline: none;`
`caption`	<caption>	`display: table-caption;` `text-align: center;`
`cite`	<cite>	`font-style: italic;`
`code`	<code>	`font-family: monospace;`
`col`	<col>	`display: table-column;`
`colgroup`	<colgroup>	`display: table-column-group`
`datalist`	<datalist>	`display: none;`
`dd`	<dd>	`display: block;` `margin-left: 40px;`

del	<del>	text-decoration: line-through;
details	<details>	display: block;
dfn	<dfn>	font-style: italic;
div		display: block;
dl	<dl>	display: block; margin: 1em 0 1em 0;
dt	<dt>	display: block;
em		font-style: italic;
embed:focus	<embed>	outline: none;
fieldset	<fieldset>	display: block; margin: 0 2px; padding: 0.35em 0.75em 0.625em; border: 2px groove (*and internal value*);
figcaption	<figcaption>	display: block;
figure	<figure>	display: block; margin: 1em 40px;
footer	<footer>	display: block;
form	<form>	display: block; margin-top: 0em;
h1	<h1>	display: block; font-size: 2em; margin: 0.67em 0; font-weight: bold;
h2	<h2>	display: block; font-size: 1.5em; margin: 0.83em 0; font-weight: bold;
h3	<h3>	display: block; font-size: 1.17em; margin: 1em 0; font-weight: bold;
h4	<h4>	display: block; margin: 1.33em 0; font-weight: bold;
h5	<h5>	display: block; font-size: 0.83em; margin-top: 1.67em 0; font-weight: bold;
h6	<h6>	display: block; font-size: 0.67em; margin-top: 2.33em 0; font-weight: bold;
header	<header>	display: block;
hr	<hr>	display: block; margin: 0.5em auto; border-style: inset; border-width: 1px;
html	<html>	display: block;
html:focus	<html>	outline: none;
i	<i>	font-style: italic;
iframe:focus	<iframe>	outline: none;
iframe[seamless]	<iframe>	display: block;
img	<img>	display: inline-block;

ins	<ins>	text-decoration: underline;
kbd	<kbd>	font-family: monospace;
label	<label>	cursor: default;
legend	<legend>	display: block; padding: 0 2px; border: none;
li		display: list-item;
link	<link>	display: none;
map	<map>	display: inline;
mark	<mark>	background-color: yellow; color: black;
menu	<menu>	display: block; list-style-type: disc; margin: 1em 0; padding-left: 40px;
nav	<nav>	display: block;
object:focus	<object>	outline: none;
ol		display: block; list-style-type: decimal; margin: 1em 0; padding-left: 40px;
output	<output>	display: inline;
p	<p>	display: block; margin: 1em 0;
param	<param>	display: none;
pre	<pre>	display: block; font-family: monospace; white-space: pre; margin: 1em 0;
q	<q>	display: inline;
q::before	<q>	content: open-quote;
q::after	<q>	content: close-quote;
rt	<rt>	line-height: normal;
s	<s>	text-decoration: line-through;
samp	<samp>	font-family: monospace;
script	<script>	display: none;
section	<section>	display: block;
small	<small>	font-size: smaller;
strong		font-weight: bold;
style	<style>	display: none;
sub	<sub>	vertical-align: sub; font-size: smaller;
summary	<summary>	display: block;
sup	<sup>	vertical-align: super; font-size: smaller;
table	<table>	display: table; border-collapse: separate; border-spacing: 2px; border-color: gray;
tbody	<tbody>	display: table-row-group; vertical-align: middle; border-color: inherit;

td	<td>	display: table-cell; vertical-align: inherit;
tfoot	<tfoot>	display: table-footer-group; vertical-align: middle; border-color: inherit;
th	<th>	display: table-cell; vertical-align: inherit; font-weight: bold; text-align: center;
thead	<thead>	display: table-header-group; vertical-align: middle; border-color: inherit;
title	<title>	display: none;
tr	<tr>	display: table-row; vertical-align: inherit; border-color: inherit;
u	<u>	text-decoration: underline;
ul		display: block; list-style-type: disc; margin: 1em 0; padding-left: 40px;
var	<var>	font-style: italic;

Appendix M: HTML Special Characters

[HTMLSpecialCharacters]

HTML markup includes some for special characters. The most frequently used are characters that if they appeared directly would be understood by HTML to relate to markup (see HTMLTutorialSpecialCharacters for further details).

Each of these markups is preceded by an ampersand, i.e. "&", followed by the markup name, followed by a semicolon, i.e. ";":

HTML name	character	name	Unicode hex code (decimal)
amp	&	ampersand	U+0026 (38)
quot	"	quotation mark	U+0022 (34)
apos	'	apostrophe	U+0027 (39)
lt	<	less than sign	U+003C (60)
gt	>	greater than sign	U+003E (62)
nbsp		non-breaking space	U+00A0 (160)
iexcl	¡	inverted exclamation mark	U+00A1 (161)
cent	¢	cent sign	U+00A2 (162)
pound	£	pound sign	U+00A3 (163)
curren	¤	currency sign	U+00A4 (164)
yen	¥	yen (yuan) sign	U+00A5 (165)
brvbar	¦	broken vertical bar	U+00A6 (166)
sect	§	section sign	U+00A7 (167)
uml	¨	diaeresis(or umlaut)	U+00A8 (168)
copy	©	copyright symbol	U+00A9 (169)
ordf	ª	feminine ordinal indicator	U+00AA (170)
laquo	«	left-pointing double angle quotation mark	U+00AB (171)
not	¬	not sign	U+00AC (172)
shy		soft (discretionary) hyphen	U+00AD (173)
reg	®	registered trademark symbol	U+00AE (174)
macr	¯	macron	U+00AF (175)
deg	°	degree symbol	U+00B0 (176)
plusmn	±	plus-minus sign	U+00B1 (177)
sup2	²	superscript two	U+00B2 (178)
sup3	³	superscript three	U+00B3 (179)
acute	´	acute accent	U+00B4 (180)
micro	µ	micro sign	U+00B5 (181)
para	¶	paragraph sign	U+00B6 (182)
middot	·	middle dot	U+00B7 (183)
cedil	¸	cedilla	U+00B8 (184)
sup1	¹	superscript one	U+00B9 (185)
ordm	º	masculine ordinal indicator	U+00BA (186)

raquo	»	right-pointing double angle quotation mark	U+00BB (187)
frac14	¼	fraction one quarter	U+00BC (188)
frac12	½	fraction one half	U+00BD (189)
frac34	¾	fraction three quarters	U+00BE (190)
iquest	¿	inverted question mark	U+00BF (191)
Agrave	À	Latin capital letter A with grave accent	U+00C0 (192)
Aacute	Á	Latin capital letter A with acute accent	U+00C1 (193)
Acirc	Â	Latin capital letter A with circumflex	U+00C2 (194)
Atilde	Ã	Latin capital letter A with tilde	U+00C3 (195)
Auml	Ä	Latin capital letter A with diaeresis	U+00C4 (196)
Aring	Å	Latin capital letter A with ring above	U+00C5 (197)
AElig	Æ	Latin capital letter AE	U+00C6 (198)
Ccedil	Ç	Latin capital letter C with cedilla	U+00C7 (199)
Egrave	È	Latin capital letter E with grave accent	U+00C8 (200)
Eacute	É	Latin capital letter E with acute accent	U+00C9 (201)
Ecirc	Ê	Latin capital letter E with circumflex	U+00CA (202)
Euml	Ë	Latin capital letter E with diaeresis	U+00CB (203)
Igrave	Ì	Latin capital letter I with grave accent	U+00CC (204)
Iacute	Í	Latin capital letter I with acute accent	U+00CD (205)
Icirc	Î	Latin capital letter I with circumflex	U+00CE (206)
Iuml	Ï	Latin capital letter I with diaeresis	U+00CF (207)
ETH	Ð	Latin capital letter Eth	U+00D0 (208)
Ntilde	Ñ	Latin capital letter N with tilde	U+00D1 (209)
Ograve	Ò	Latin capital letter O with grave accent	U+00D2 (210)
Oacute	Ó	Latin capital letter O with acute accent	U+00D3 (211)
Ocirc	Ó	Latin capital letter O with circumflex	U+00D4 (212)
Otilde	Õ	Latin capital letter O with tilde	U+00D5 (213)
Ouml	Ö	Latin capital letter O with diaeresis	U+00D6 (214)
times	×	multiplication sign	U+00D7 (215)
Oslash	Ø	Latin capital letter O with stroke (Latin capital letter O slash)	U+00D8 (216)
Ugrave	Ù	Latin capital letter U with grave accent	U+00D9 (217)
Uacute	Ú	Latin capital letter U with acute accent	U+00DA (218)
Ucirc	Û	Latin capital letter U with circumflex	U+00DB (219)
Uuml	Ü	Latin capital letter U with diaeresis	U+00DC (220)
Yacute	Ý	Latin capital letter Y with acute accent	U+00DD (221)
THORN	Þ	Latin capital letter THORN	U+00DE (222)
szlig	ß	Latin small letter sharp s, i.e. German Eszett	U+00DF (223)
agrave	à	Latin small letter a with grave accent	U+00E0 (224)
aacute	á	Latin small letter a with acute accent	U+00E1 (225)
acirc	â	Latin small letter a with circumflex	U+00E2 (226)
atilde	ã	Latin small letter a with tilde	U+00E3 (227)

auml	ã	Latin small letter a with diaeresis	U+00E4 (228)
aring	å	Latin small letter a with ring above	U+00E5 (229)
aelig	æ	Latin small letter ae	U+00E6 (230)
ccedil	ç	Latin small letter c with cedilla	U+00E7 (231)
egrave	è	Latin small letter e with grave accent	U+00E8 (232)
eacute	é	Latin small letter e with acute accent	U+00E9 (233)
ecirc	ê	Latin small letter e with circumflex	U+00EA (234)
euml	ë	Latin small letter e with diaeresis	U+00EB (235)
igrave	ì	Latin small letter i with grave accent	U+00EC (236)
iacute	í	Latin small letter i with acute accent	U+00ED (237)
icirc	î	Latin small letter i with circumflex	U+00EE (238)
iuml	ï	Latin small letter i with diaeresis	U+00EF (239)
eth	ð	Latin small letter eth	U+00F0 (240)
ntilde	ñ	Latin small letter n with tilde	U+00F1 (241)
ograve	ò	Latin small letter o with grave accent	U+00F2 (242)
oacute	ó	Latin small letter o with acute accent	U+00F3 (243)
ocirc	ô	Latin small letter o with circumflex	U+00F4 (244)
otilde	õ	Latin small letter o with tilde	U+00F5 (245)
ouml	ö	Latin small letter o with diaeresis	U+00F6 (246)
divide	÷	division sign (obelus)	U+00F7 (247)
oslash	ø	Latin small letter o with stroke (Latin small letter o slash)	U+00F8 (248)
ugrave	ù	Latin small letter u with grave accent	U+00F9 (249)
uacute	ú	Latin small letter u with acute accent	U+00FA (250)
ucirc	û	Latin small letter u with circumflex	U+00FB (251)
uuml	ü	Latin small letter u with diaeresis	U+00FC (252)
yacute	ý	Latin small letter y with acute accent	U+00FD (253)
thorn	þ	Latin small letter thorn	U+00FE (254)
yuml	ÿ	Latin small letter y with diaeresis	U+00FF (255)
OElig	Œ	Latin capital ligature oe	U+0152 (338)
oelig	œ	Latin small ligature oe	U+0153 (339)
Scaron	Š	Latin capital letter s with caron	U+0160 (352)
scaron	š	Latin small letter s with caron	U+0161 (353)
Yuml	Ÿ	Latin capital letter y with diaeresis	U+0178 (376)
fnof	ƒ	Latin small letter f with hook (function, florin)	U+0192 (402)
circ	ˆ	modifier letter circumflex accent	U+02C6 (710)
tilde	˜	small tilde	U+02DC (732)
Alpha	Α	Greek capital letter Alpha	U+0391 (913)
Beta	Β	Greek capital letter Beta	U+0392 (914)
Gamma	Γ	Greek capital letter Gamma	U+0393 (915)
Delta	Δ	Greek capital letter Delta	U+0394 (916)

Epsilon	Ε	Greek capital letter Epsilon	U+0395 (917)
Zeta	Ζ	Greek capital letter Zeta	U+0396 (918)
Eta	Η	Greek capital letter Eta	U+0397 (919)
Theta	Θ	Greek capital letter Theta	U+0398 (920)
Iota	Ι	Greek capital letter Iota	U+0399 (921)
Kappa	Κ	Greek capital letter Kappa	U+039A (922)
Lambda	Λ	Greek capital letter Lambda	U+039B (923)
Mu	Μ	Greek capital letter Mu	U+039C (924)
Nu	Ν	Greek capital letter Nu	U+039D (925)
Xi	Ξ	Greek capital letter Xi	U+039E (926)
Omicron	Ο	Greek capital letter Omicron	U+039F (927)
Pi	Π	Greek capital letter Pi	U+03A0 (928)
Rho	Ρ	Greek capital letter Rho	U+03A1 (929)
Sigma	Σ	Greek capital letter Sigma	U+03A3 (931)
Tau	Τ	Greek capital letter Tau	U+03A4 (932)
Upsilon	Υ	Greek capital letter Upsilon	U+03A5 (933)
Phi	Φ	Greek capital letter Phi	U+03A6 (934)
Chi	Χ	Greek capital letter Chi	U+03A7 (935)
Psi	Ψ	Greek capital letter Psi	U+03A8 (936)
Omega	Ω	Greek capital letter Omega	U+03A9 (937)
alpha	α	Greek small letter alpha	U+03B1 (945)
beta	β	Greek small letter beta	U+03B2 (946)
gamma	γ	Greek small letter gamma	U+03B3 (947)
delta	δ	Greek small letter delta	U+03B4 (948)
epsilon	ε	Greek small letter epsilon	U+03B5 (949)
zeta	ζ	Greek small letter zeta	U+03B6 (950)
eta	η	Greek small letter eta	U+03B7 (951)
theta	θ	Greek small letter theta	U+03B8 (952)
iota	ι	Greek small letter iota	U+03B9 (953)
kappa	κ	Greek small letter kappa	U+03BA (954)
lambda	λ	Greek small letter lambda	U+03BB (955)
mu	μ	Greek small letter mu	U+03BC (956)
nu	ν	Greek small letter nu	U+03BD (957)
xi	ξ	Greek small letter xi	U+03BE (958)
omicron	ο	Greek small letter omicron	U+03BF (959)
pi	π	Greek small letter pi	U+03C0 (960)
rho	ρ	Greek small letter rho	U+03C1 (961)
sigmaf	ς	Greek small letter final sigma	U+03C2 (962)
sigma	σ	Greek small letter sigma	U+03C3 (963)
tau	τ	Greek small letter tau	U+03C4 (964)
upsilon	υ	Greek small letter upsilon	U+03C5 (965)
phi	φ	Greek small letter phi	U+03C6 (966)

chi	χ	Greek small letter chi	U+03C7 (967)
psi	ψ	Greek small letter psi	U+03C8 (968)
omega	ω	Greek small letter omega	U+03C9 (969)
thetasym	ϑ	Greek theta symbol	U+03D1 (977)
upsih	ϒ	Greek Upsilon with hook symbol	U+03D2 (978)
piv	ϖ	Greek pi symbol	U+03D6 (982)
ensp		en space[d]	U+2002 (8194)
emsp		em space[d]	U+2003 (8195)
thinsp		thin space[d]	U+2009 (8201)
zwnj		zero-width non-joiner	U+200C (8204)
zwj		zero-width joiner	U+200D (8205)
lrm		left-to-right mark	U+200E (8206)
rlm		right-to-left mark	U+200F (8207)
ndash	–	en dash	U+2013 (8211)
mdash	—	em dash	U+2014 (8212)
lsquo	‘	left single quotation mark	U+2018 (8216)
rsquo	’	right single quotation mark	U+2019 (8217)
sbquo	‚	single low-9 quotation mark	U+201A (8218)
ldquo	“	left double quotation mark	U+201C (8220)
rdquo	”	right double quotation mark	U+201D (8221)
bdquo	„	double low-9 quotation mark	U+201E (8222)
dagger	†	dagger, obelisk	U+2020 (8224)
Dagger	‡	double dagger, double obelisk	U+2021 (8225)
bull	•	bullet (black small circle)	U+2022 (8226)
hellip	…	horizontal ellipsis (three dot leader)	U+2026 (8230)
permil	‰	per mille sign	U+2030 (8240)
prime	′	prime (minutes, feet)	U+2032 (8242)
Prime	″	double prime (seconds, inches)	U+2033 (8243)
lsaquo	‹	single left-pointing angle quotation mark	U+2039 (8249)
rsaquo	›	single right-pointing angle quotation mark	U+203A (8250)
oline	‾	overline (spacing overscore)	U+203E (8254)
frasl	⁄	fraction slash (solidus)	U+2044 (8260)
euro	€	euro sign	U+20AC (8364)
image	ℑ	black-letter capital I (imaginary part)	U+2111 (8465)
weierp	℘	script capital P (power set, Weierstrass p)	U+2118 (8472)
real	ℜ	black-letter capital R (real part symbol)	U+211C (8476)
trade	™	trademark symbol	U+2122 (8482)
alefsym	ℵ	alef symbol (first transfinite cardinal)	U+2135 (8501)
larr	←	leftwards arrow	U+2190 (8592)
uarr	↑	upwards arrow	U+2191 (8593)
rarr	→	rightwards arrow	U+2192 (8594)
darr	↓	downwards arrow	U+2193 (8595)

harr	↔	left right arrow	U+2194 (8596)
crarr	↵	downwards arrow with corner leftwards (carriage return)	U+21B5 (8629)
lArr	⇐	leftwards double arrow	U+21D0 (8656)
uArr	⇑	upwards double arrow	U+21D1 (8657)
rArr	⇒	rightwards double arrow	U+21D2 (8658)
dArr	⇓	downwards double arrow	U+21D3 (8659)
hArr	⇔	left right double arrow	U+21D4 (8660)
forall	∀	for all	U+2200 (8704)
part	∂	partial differential	U+2202 (8706)
exist	∃	there exists	U+2203 (8707)
empty	∅	empty set (null set)	U+2205 (8709)
nabla	∇	del or nabla (vector differential operator)	U+2207 (8711)
isin	∈	element of	U+2208 (8712)
notin	∉	not an element of	U+2209 (8713)
ni	∋	contains as member	U+220B (8715)
prod	∏	n-ary product (product sign)	U+220F (8719)
sum	∑	n-ary summation	U+2211 (8721)
minus	−	minus sign	U+2212 (8722)
lowast	∗	asterisk operator	U+2217 (8727)
radic	√	square root (radical sign)	U+221A (8730)
prop	∝	proportional to	U+221D (8733)
infin	∞	infinity	U+221E (8734)
ang	∠	angle	U+2220 (8736)
and	∧	logical and (wedge)	U+2227 (8743)
or	∨	logical or (vee)	U+2228 (8744)
cap	∩	intersection (cap)	U+2229 (8745)
cup	∪	union (cup)	U+222A (8746)
int	∫	integral	U+222B (8747)
there4	∴	therefore sign	U+2234 (8756)
sim	∼	tilde operator (varies with, similar to)	U+223C (8764)
cong	≅	congruent to	U+2245 (8773)
asymp	≈	almost equal to (asymptotic to)	U+2248 (8776)
ne	≠	not equal to	U+2260 (8800)
equiv	≡	identical to or 'equivalent to'	U+2261 (8801)
le	≤	less-than or equal to	U+2264 (8804)
ge	≥	greater-than or equal to	U+2265 (8805)
sub	⊂	subset of	U+2282 (8834)
sup	⊃	superset of	U+2283 (8835)
nsub	⊄	not a subset of	U+2284 (8836)
sube	⊆	subset of or equal to	U+2286 (8838)
supe	⊇	superset of or equal to	U+2287 (8839)

`oplus`	⊕	circled plus (direct sum)	U+2295 (8853)
`otimes`	⊗	circled times (vector product)	U+2297 (8855)
`perp`	⊥	up tack (orthogonal to, perpendicular)	U+22A5 (8869)
`sdot`	⋅	dot operator	U+22C5 (8901)
`lceil`	⌈	left ceiling	U+2308 (8968)
`rceil`	⌉	right ceiling	U+2309 (8969)
`lfloor`	⌊	left floor	U+230A (8970)
`rfloor`	⌋	right floor	U+230B (8971)
`lang`	⟨	left-pointing angle bracket (bra)	U+2329 (9001)
`rang`	⟩	right-pointing angle bracket (ket)	U+232A (9002)
`loz`	◊	lozenge	U+25CA (9674)
`spades`	♠	spade suit	U+2660 (9824)
`clubs`	♣	club suit (shamrock)	U+2663 (9827)
`hearts`	♥	heart suit (valentine)	U+2665 (9829)
`diams`	♦	diamond suit	U+2666 (9830)

Note: an ampersand "`&`" will usually display if it is not part of a special character, but it is better to use HTML markup for it, i.e. `&`.

Appendix N: Markup Languages

[Nematrian website page: HTMLMarkupLanguages, © Nematrian 2017]

HTML is the main 'markup' language used for web pages and web applications. By a (digital) markup language we mean a way of creating and interpreting a digital document in which the document contains tags (and their attributes) that the software rendering the document interprets in a specific way (but with the tags themselves and their attributes not typically directly appearing in the output transmitted to the user). In what follows we will describe how this concept works with documents concentrating on textual output, although the same concepts are also applicable to documents containing other types of material (such as pictures or sounds).

There are many different mark-up languages used in different contexts. For example, LaTeX (and TeX, the underlying mark-language on which LaTeX is based) is a tool for preparing mathematically orientated documents. It uses the backslash character ("\") and braces ("{" and "}") to tell the software rendering the document that relevant text needs to be interpreted in a specific manner. Text of the form "E &= \frac{mc^2}{\sqrt{1-\frac{v^2}{c^2}}}" is rendered by a TeX viewer roughly along the lines of the following:

$$E = \frac{mc^2}{\sqrt{1-\frac{v^2}{c^2}}}$$

Here the "\frac{numerator}{denominator}" tells the software to render the text formed by the numerator and the denominator as a fraction, and \sqrt{argument} tells the software to render the text formed by the argument as a square root. Markup can be nested.

Certain features are shared by virtually all digital mark-up languages, including HTML. These are:

(a) The mark-up language needs to be specified and interpreted in a consistent fashion. This is harder to arrange than it looks for languages that develop through time, since the equivalent of different dialects can then be created.

At the time of writing, the latest formally adopted version of HTML is HTML 4.01 although the World Wide Web Consortium (W3C) issued HTML 5 as a formal recommendation in October 2014 and has also developed a parallel XML based language, XHTML 5.1. XML stands for "eXtensible Mark-up Language". Most leading browsers will interpret an HTML document using HTML 5 conventions, but some older browsers may not. Modern browsers can be instructed to use older versions of the language if necessary by including a suitable document-level tag. HTML 4 itself comes in three different versions, i.e. Strict, Transitional and Frameset. These loosely-speaking correspond to how closely the document adheres to the specific requirements of HTML 4.

(b) The language generally needs to be able to nest tags within other tags. This requires the language to have the concept of opening a tag and then closing it, with the text in-between the opening and closing elements being interpreted in a specific manner. With TeX, the nesting process makes use of open and close braces ("{" and "}" respectively). With HTML, tags (more commonly called 'elements') generally take a form akin to `<xxx> … </xxx>`, where the `<xxx>` opens the tag, the `</xxx>` closes the tag and the `xxx` represents the type of tag involved. More sophisticated tags take the form:

```
<xxx yyy> … </xxx>
```

where the `yyy` defines the tag's attributes, i.e. provides added information on (i.e. attributes for) the element / tag.

For example, any text in a webpage between an opening `<script>` tag and the corresponding closing `</script>` is generally interpreted as JavaScript code. Any text between an opening `<a>` and a closing `</a>` is the text used when rendering a hyperlink. The address of the document to which the hyperlink points is included as an element attribute, e.g. the full tag might involve:

```
<a href="http://www.nematrian.com/Introduction.aspx">
Introduction to Nematrian website </a>).
```

Some mark-up languages such as XML require all opened tags to be explicitly closed, e.g. with any `<x>` ultimately closed by a `</x>` (or in XML it is possible to open and close a tag at the same time, using a format such as `<x />`). Others, like HTML, do not require this convention, if the tag never contains anything. For example, in HTML the tag `<br>` means insert a carriage break, i.e. start a new line, and does not need to be followed by a `</br>`.

Appendix O: JavaScript Statements (and Directives): Reserved Words

[Nematrian website page: JavaScriptStatements, © Nematrian 2020]

JavaScript statements identify instructions that are executed by the web browser. A summary of JavaScript statements is given here.

A list of the main statement reserved words recognised by JavaScript is shown here:

Statement	Description	More
`break`	Exits a switch or loop	Here
`class`	Defines a class	Here
`const`	Declares a constant	Here
`continue`	Breaks an iteration (in a loop) if a specific condition occurs, moving on to the next iteration	Here
`debugger`	Stops execution of JavaScript and calls the debugging capability if available	
`do … while`	Executes a block of statements, then repeats the block while the condition remains true	Here
`for`	Marks a block of statements to be executed whilst a condition is true	Here
`for … in`	Marks a block of statements to be executed for each element of an object	Here
`function`	Declares a function	Here
`if … else if … else`	Marks a block of statements to be executed depending on a condition	Here
`let`	Declares a variable (with block scope)	Here
`return`	Stops execution of a function and returns a value from that function	Here
`switch`	Marks a block of statements to be executed depending on different cases	Here
`this`	Used to access the properties and methods of the object that has defined the relevant object	Here
`throw`	Throws an error object as part of implementing error handling	Here
`try … catch … finally`	Implements error handling	Here
`var`	Declares a variable (with global or function scope)	Here
`while`	Identifies block of statements that is repeatedly executed while a condition is true	Here

Most JavaScript programs contain many statements, which are executed one by one in the order in which they are written except when statement flow control is adjusted as above.

Some reserved words are in practice limited to `class` definitions, including:

Statement	Description	More
`class`	Defines a class	Here
`extends`	Allows one class to inherit the methods and properties of another class	Here
`get`	Used to get a class property	Here

set	Used to set a class property	Here
static	Used to define a method on the class itself rather than on individual instances of the class	Here
super	Breaks an iteration (in a loop) if a specific condition occurs, moving on to the next iteration	Here

The this reserved word, see here, is often also used in classes, although it has more general application. Code in classes automatically needs to be in strict mode, other code can be forced to be in strict mode using the use strict directive.

Set out below is a list of JavaScript reserved words (which cannot be used as variables, labels or function names):

abstract**	arguments	await*	boolean**
break	byte**	case	catch
char**	class*	const	continue
debugger	default	delete	do
double**	else	enum*	eval
export*	extends*	false	final**
finally	float**	for	function
goto**	if	implements	import*
in	instanceof	int**	interface
let*	long**	native**	new
null	package	private	protected
public	return	short**	static
super*	switch	synchronized**	this
throw	throws**	transient**	true
try	typeof	var	void
volatile**	while	with	yield

* Are new in ECMAScript 5 and 6
** Removed from the ECMAScript 5/6 standard, but it is recommended not to use them, because at the time of writing ECMAScript 5/6 was not fully supported by all browsers.

Individual statement reserved words:

break

[JavaScriptStatementBreak]

In JavaScript, the break statement breaks an iteration (in a loop) if a specific condition occurs, moving to the next statement after the entire iteration (or when used inside a switch statement it moves on to the next statement after then entire switch statement).

class

[JavaScriptStatementClass]

In JavaScript, the class statement defines a class. This is technically a special type of JavaScript function that adds additional object-orientated characteristics to JavaScript. It was introduced in

ECMAScript 2015. Once a class has been defined, other variables can be created that are instances of this class, with methods and properties as defined by the class.

The `class` keyword is used to create a class. A specific method, the `constructor` method, is automatically called each time an object (i.e. an instance of the class) is initialised. The properties of the object instance can then be initialised using the `this` statement. If you do not include a constructor method then JavaScript will add an invisible and empty constructor method. (Other) methods can then be added (without needing a function statement). Static methods, defined using the `static` statement, are defined on the class itself and not on any instance of the class.

Classes can inherit properties and methods from other classes using the extends keyword. The super method can be used in the constructor method to access the properties and methods of the parent.

In object-orientated computer languages classes also have 'get' and 'set' methods that allow properties within an object of a given class to be accessed or set by statements manipulating the object (and, sometimes, when doing so for other intra-class manipulations to take place). In JavaScript this functionality is achieved using the get and set statements. The names of getter/setter methods cannot be the same as the name of the property manipulated by the method (programmers often use an underscore character _ before the property name to differentiate the getter/setter from the actual property). To use the getter/setter methods you use the same syntax as when you set a property value, without parentheses.

const

[JavaScriptStatementConst]

In JavaScript, the `const` statement declares a constant. This is akin to a variable defined using a `var` statement, see here, or a `let` statement, see here, but with the variable being unable to be changed thereafter. Both the `const` and the `var` statements were introduced by ECMAScript 2015.

It is generally considered good practice to define variables as constants if they are not going to change, as it reduces the risk of them being accidentally overwritten.

continue

[JavaScriptStatementContinue]

In JavaScript, the `continue` statement breaks an iteration (in a loop) if a specific condition occurs, moving on to the next iteration.

do … while

[JavaScriptStatementDoWhile]

In JavaScript, the `do … while` statement executes a block of statements, then repeats the block while the condition remains true.

extends

[JavaScriptStatementExtends]

In JavaScript, the `extends` keyword is used within the definition of one class to inherit the properties and methods of another class.

for

[JavaScriptStatementFor]

In JavaScript, the `for` statement identifies a block of statements that are to be executed whilst a condition is true (and a `break` or to some extent a `continue` statement have not been triggered).

For example, statements such as:

```
var sumi = 0;
var i;
for (i = 1; i <= 5; i++) {sumi = sumi + i}
```

will loop through from `i=1` to `i=5` and calculate the sum (i.e. 1+2+3+4+5 = 15)

for … in

[JavaScriptStatementForIn]

In JavaScript, the `for … in` statement marks a block of statements to be executed for each element of an object (or array, although with an array it is typically better to use the for statement and to iterate over the indices of the array elements).

function

[JavaScriptStatementFunction]

In JavaScript, the `function` statement declares a function (akin to a subroutine or procedure in some other programming languages), i.e. a set of statements that can be executed (which can return a result) from elsewhere within the code.

get

[JavaScriptStatementGet]

In JavaScript, the `get` statement is used to get a property of an instance of a given class. Within the statements associated with the `get` statement it is also possible to manipulate the class whilst the getting is occurring.

if … else if … else

[JavaScriptStatementIf]

In JavaScript, the `if` statement marks a block of statements to be executed depending on a condition. There are three types of `if` statement:

```
(1) if (…) {…}
(2) if (…) {…} else {…}
(3) if (…) {…} else if (…) {…}
```

The expression within the (first) normal brackets, i.e. within matching "(" and ")", should evaluate to a Boolean (i.e. be a condition). If this condition is true then the next code block (within curly brackets) will be executed, whilst if this condition is false and there is an `else` statement then the code in the second curly brackets in (2) would be executed. The `else if` variant in (3) allows further (potentially several nested) conditions to be included and can include a final `else` block as per (2).

There is one further type of conditional statement, the switch statement, which is (typically) used when there are multiple cases each of which would trigger execution of different code blocks.

let

[JavaScriptStatementLet]

In JavaScript, the `let` statement declares a variable with 'Block Scope'.

Prior to ECMAScript 2015, variables could only be declared using the `var` statement, see here. The `var` statement defined the variable either globally (outside any function), i.e. with 'Global Scope', or locally within a function, i.e. with 'Function Scope'. In contrast, a `let` statement defines the variable within a given block, i.e. within a matching `{` and `}`.

The variable value with 'Block Scope' inside the block cannot be accessed from outside the block. If the same variable has apparently been declared before the block starts, then the variable returns to having that value after the block finishes.

return

[JavaScriptStatementReturn]

In JavaScript, the `return` statement stops execution of a function and returns a value from that function.

set

[JavaScriptStatementSet]

In JavaScript, the `set` statement is used to set a property of an instance of a given class. Within the statements associated with the `set` statement it is also possible to manipulate the class whilst the setting is occurring.

static

[JavaScriptStatementStatic]

In JavaScript, the `static` statement is used to define a method on a class itself, rather than on any instances of the class.

strict mode and the “use strict” directive

[JavaScriptStatementStrictMode]

The `"use strict"` statement (more precisely a ‘directive’ or literal expression) indicates the relevant part of the code should be executed in “strict mode”. In strict mode, for example, you cannot use undeclared variables. It was introduced in ECMAScript 2015.

Strict mode is declared by adding `"use strict"` to the beginning of the script or function to which it applies:

- If it is declared at the beginning of a script it has global scope (i.e. applies to all code in the script
- If it is declared inside a function (at its start) then it has local scope (i.e. applies only to code inside the function

Its syntax is designed to be compatible with older versions of JavaScript. Compiling a numerical literal (e.g. `1+2`) or a string literal `"hello"` simply compiles to a non-existent variable, so has no practical impact, so an ‘assignment’ like `"use strict"` is in effect ignored by older versions of JavaScript.

The main advantage of strict mode is that it makes it easier to write fault-free JavaScript, since it makes it easier to pick up errors that are the result of otherwise bad syntax. Actions that are not allowed in strict mode include:

- Using a variable without declaring it
- Using an object without declaring it
- Deleting a variable, object or function
- Duplicating a parameter name
- Octal numeric literals and Octal escape characters
- Writing to a read-only property
- Writing to a get-only property
- Deleting an undeletable property
- Using the `with` statement
- Using `eval()` to create variables in the scope from which it is called

ECMAScript 2015 also prohibited a range of keywords, some then still to be finalised, from being used as variable names including e.g.: `argument`, `eval` , `implements`, `interface`, `let`, `package`, `private`, `protected`, `public`, `static`, `yield`

The `this` keyword also behaves differently in strict mode. The `this` keyword refers to the object that called the function. If this object is not specified then in strict mode this will return undefined, whilst in normal (i.e. not strict) mode it will return the global object (i.e. the window).

Code within classes is automatically deemed to be written in strict mode.

super

[JavaScriptStatementSuper]

In JavaScript, the `super` method is used within a class that has been extended from another class to access the properties and methods of the parent class.

switch {case … case … default …}

[JavaScriptStatementSwitch]

In JavaScript, the `switch` statement marks a block of statements to be executed depending on different cases that an expression provided with the `switch` statement takes. The statement takes e.g. the following format:

```
var x;
switch(expression) {
  case 1:
    x = 10;
    break;
  case 3:
    x = 20;
    break;
  default:
    x = 5;
}
```

In the above the *expression* would be evaluated. If its value was `1` then the code following the `case 1` statement would be executed, if it was `2` then the code following the `case 3` statement would be executed, and if it was neither then the code immediately after the (optional) `default` statement would be executed.

The format of this statement differs from e.g. the `select … case … case else` statement in Visual Basic. The VB `select` statement results in only the code immediately following the relevant `case` or `case else` statement being executed. In contrast, with JavaScript, all the code from the relevant `case` statement to the end of the `switch` expression block is executed, until a `break` statement is reached. So, in the above, if the first break statement was commented out then `x` would become `20` if expression was either `1` or `3` (if it was one then it would be set to `10` but then subsequently set to `20`).

If *expression* evaluates to a Boolean (i.e. true or false) then it is more common to use the if (or variants such as if … else) statement.

this

[JavaScriptStatementThis]

In JavaScript, the `this` element allows you to access the properties and methods of the object that defines whatever is being considered at the time. A common use is to access the properties of an instance of a class. The `this` element then refers to these properties, which can then be manipulated in the constructor, get and set and other methods of the class.

throw

[JavaScriptStatementThrow]

In JavaScript, the `throw` statement (often used in conjunction with the try … catch statement) implements error handling. It throws an exception (technically an Error object).

The exception can be specified as just some text (e.g. `throw "Error"`) or a number (e.g. `throw 100`) or more generally an error object, e.g. `throw new Error(100,"Error")`.

try … catch … finally

[JavaScriptStatementTry]

In JavaScript, the `try … catch … finally` statement (often used in conjunction with the throw statement) implements error handling. The statement takes e.g. the following format:

```
try {
  code1
}
catch(e) {
  code2
}
finally {
  code3
}
```

In the above, JavaScript will first try to execute *code1*. If an error occurs, e.g. the code cannot be understood or is misspelt and the (optional) catch statement is present then instead of just stopping (which would be the usual response to an error in the absence of a `try` statement) it moves to *code2*. Regardless of the try / catch result, if there is an (optional) `finally` statement it will then execute *code3*. The type of error thrown (which can be system specified or specified by the developer using the throw statement is available through *e*, which is the name used in the code to specify the local Error object identifying the error.

Note: the `catch` and `finally` statement components are both optional, but you typically need to include at least one of them when using a try statement.

Error objects have two intrinsic properties, the `.message` property which contains a description of the error and the `.number` property which contains the error number of the error. Note, some browser suppliers e.g. Microsoft have additional non-standard properties such as `.description`, which seems otherwise to be the same as `.message` but will not be recognised by non-Microsoft browsers (so should be avoided if users are likely to use other browsers to view the relevant webpage).

Other modern object-orientated programming languages such as Visual Basic also typically now include structured error handling like the above, but potentially with different forms of error object and with the error object potentially more simply handling errors triggered within function calls.

var

[JavaScriptStatementVar]

In JavaScript, the `var` statement declares a variable.

while

[JavaScriptStatementWhile]

In JavaScript, the `while` statement identifies block of statements that is repeatedly executed while a condition is true.

Appendix P: JavaScript String Variables

[JavaScriptTutorialStrings]

JavaScript strings consist of a series of consecutive characters, e.g.

```
var x = "Cat";
```

A string technically consists of a series (an 'array', except that a JavaScript array is a specific type of variable) of characters, which is zero-indexed. So, if we assigned `x` the value of `"Cat"` then `x[0]` would be `"C"`, `x[1]` would be `"a"`, etc.

Strings support the following properties and methods. Some of these involve regular expressions.

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`length`	Returns length of string	Here
`prototype`	Allows author to add properties and methods to an object	Here

Methods:

Method	Description	More
`charAt()`	Returns the character at specified index position (note strings in JavaScript are zero index based, so the first character is at position zero)	Here
`charCodeAt()`	Returns the Unicode character code of the character at specified index position (note strings in JavaScript are zero index based, so the first character is at position zero)	Here
`concat()`	Returns the result of joining two or more strings together	Here
`endsWith()`	Returns true if the string ends with a specified string, otherwise returns false	Here
`fromCharCode()`	Returns the string corresponding to a specified Unicode character	Here
`includes()`	Returns true if the string contains a specified string, otherwise returns false	Here
`indexOf()`	Returns the position of the first occurrence of a specified string in the string	Here
`lastIndexOf()`	Returns the position of the last occurrence of a specified string in the string	Here
`localeCompare()`	Returns a number which is -1 if string is before specified string in sort order, 0 if they are the same and +1 if string is after specified string in sort order	Here
`match()`	Searches for matches within a string versus a specified regular expression and returns these as a string array	Here
`repeat()`	Returns a string that repeats a specified string a specified number of times	Here

`replace()`	Searches for matches within a string versus a specified value (or regular expression) and returns a string in which these are replaced by another string	Here
`search()`	Searches for matches within a string versus a specified value or regular expression and returns the position of first occurrence of a match (or -1 if there is no match)	Here
`slice()`	Returns a new string formed by a part of the original string	Here
`split()`	Returns an array of substrings that are created by splitting the original string using a given delimiter	Here
`startsWith()`	Returns true if the string starts with a specified string, otherwise returns false	Here
`substr()`	Returns a substring defined by the start position and number of characters	Here
`substring()`	Returns a substring defined by the start and end position (not including the end position). If the start position is after the end position then the two are treated as reversed.	Here
`toLocaleLowerCase()`	Returns a string that is the original string converted to lower case characters, bearing in mind the language settings of the browser (so sometimes does not return the same as `toLowerCase`)	Here
`toLocaleUpperCase()`	Returns a string that is the original string converted to upper case characters, bearing in mind the language settings of the browser (so sometimes does not return the same as `toUpperCase`)	Here
`toLowerCase()`	Returns a string that is the original string converted to lower case characters	Here
`toString()`	Returns the (string) value of a string	Here
`toUpperCase`	Returns a string that is the original string converted to upper case characters	Here
`trim()`	Returns a string with whitespace (i.e. spaces) removed from start and finish of string	Here
`valueOf()`	Returns the primitive value of an object. For a string, this in effect just returns the string value of the string	Here

The Unicode character code returned by `charCodeAt()` or used as input to `fromCharCode()` is developed by the Unicode Consortium.

To handle special characters, you 'escape' the character using an escape sequence starting with a backslash character, typically followed by one of a handful of specially recognised characters (many of which were originally designed to control typewriters, so do not make much sense in HTML), or by a Unicode character of the form u followed by 4 hexadecimal characters. The following table lists a few examples of such escape sequences.

Unicode character	Alternative escape sequence (if exists)	Meaning / comment
\u005C	`\\`	Backslash, , i.e. \
\u0008	`\b`	Backspace
\u000C	`\f`	Form feed

\u000A	`\n`	Line feed (i.e. new line)
\u000D	`\r`	Carriage return
\u0009	`\t`	Horizontal tab
\u000B	`\v`	Vertical tab
\u0027	`\'`	Single quote, i.e. '
\u0022	`\"`	Double quote, i.e. "
\u0020		Space
\u00A0		Non-breaking space
\u2028		Line separator
\u2029		Paragraph separator

Single quotation marks do not seem to need to be escaped in a string which is delimited by double quotation marks and vice-versa, e.g. it is generally possible to define strings using e.g. the following, without needing to escape the quotation mark contained in the string.

```
var x = "a single quotation mark: '";
var y = 'a double quotation mark: "'
```

Some further quotation mark characters are included in the page on the CSS quotes property. Other types of escaping that work in other HTML or CSS contexts (or even in JavaScript regular expressions), e.g. using \005C rather than \u005C, do not necessarily work consistently or at all in JavaScript. So, for cross-browser compatibility it is usually desirable to use either the short escape sequence as above (if it exists) or a full Unicode escape sequence.

String properties:

length

[JavaScriptPropertyStringLength]

The `length` property (for a JavaScript string) returns the length of the string. An empty string has length 0.

It has the following syntax:

string`.length`

String methods:

charAt()

[JavaScriptMethodStringCharAt]

The `charAt()` method (when applied to a JavaScript string) returns the character at specified index position (note: strings in JavaScript are zero index based, so the first character is at position zero).

It has the following syntax with the following parameters:

string`.charAt(`*indexvalue*`)`

Parameter	Required / Optional	Description
indexvalue	Required	Integer indicating index (position) of character to return

charCodeAt()

[JavaScriptMethodStringCharCodeAt]

The `charCodeAt()` method (when applied to a JavaScript string) returns the Unicode character code of the character at specified index position (note: strings in JavaScript are zero index based, so the first character is at position zero).

It has the following syntax with the following parameters:

string.`charCodeAt(`*indexvalue*`)`

Parameter	Required / Optional	Description
indexvalue	Required	Integer indicating index (position) of character for which to return its Unicode character code

concat()

[JavaScriptMethodStringConcat]

The `concat()` method (when applied to a JavaScript string) returns the result of joining two or more strings together.

It has the following syntax with the following parameters:

string.`concat(`*string1, string2, …*`)`

Parameter	Required / Optional	Description
string1, string2, …	Required	Strings to be concatenated (joined) together

endsWith()

[JavaScriptMethodStringEndsWith]

The `endsWith()` method (when applied to a JavaScript string) returns true if the string ends with a specified string, otherwise returns false.

It has the following syntax with the following parameters:

string.`endsWith(`*searchstring, length*`)`

Parameter	Required / Optional	Description
searchstring	Required	String to be searched for
length	Optional	(Default is string.`length`). Length of string to search

fromCharCode()

[JavaScriptMethodStringFromCharCode]

The `fromCharCode()` method (when applied to the JavaScript String object) converts Unicode values into characters.

It has the following syntax with the following parameters:

`String.fromCharCode(`*n1, n2, …*`)`

Parameter	Required / Optional	Description
n1, n2, …	Required	One or more Unicode values to be converted into a string

includes()

[JavaScriptMethodStringIncludes]

The `includes()` method (when applied to a JavaScript string) returns converts Unicode values into characters.

It has the following syntax with the following parameters:

string`.includes(`*searchstring, start*`)`

Parameter	Required / Optional	Description
searchstring	Required	String to be searched for
start	Optional	(Default is `0`). Position at which to start search

indexOf()

[JavaScriptMethodStringIndexOf]

The `indexOf()` method (when applied to a JavaScript string) returns the position of the first occurrence of a specified string in the string. It is case-sensitive. It returns `-1` if the string being searched for is not found.

It has the following syntax with the following parameters:

string`.indexOf(`*searchstring, start*`)`

Parameter	Required / Optional	Description
searchstring	Required	String to be searched for
start	Optional	(Default is `0`). Position at which to start search

lastIndexOf()

[JavaScriptMethodStringLastIndexOf]

The `lastIndexOf()` method (when applied to a JavaScript String) returns the position of the last occurrence of a specified string in the string. It is case-sensitive. It returns `-1` if the string being searched for is not found.

It has the following syntax with the following parameters:

string.`lastIndexOf`(*searchstring, start*)

Parameter	Required / Optional	Description
searchstring	Required	String to be searched for (searching backwards
start	Optional	(Default is *string*.`length`). Position at which to start search

localeCompare()

[JavaScriptMethodStringLocaleCompare]

The `localeCompare()` method (when applied to a JavaScript string) number which is -1 if string is before specified (compare) string in ascending sort order, 0 if they are the same and +1 if string is after specified string in ascending sort order. It is case-sensitive.

It has the following syntax with the following parameters:

string.`localeCompare`(*comparestring*)

Parameter	Required / Optional	Description
comparestring	Required	String to be searched for

match()

[JavaScriptMethodStringMatch]

The `match()` method (when applied to a JavaScript string) searches the string for regular expression matches, and returns them as an array. If the regular expression does not include a `g` modifier (corresponding to a global search), `match` only returns the first match. If no match is found then the method returns `null`.

It has the following syntax with the following parameters:

string.`match`(*regexpression*)

Parameter	Required / Optional	Description
regexpression	Required	Regular expression being matched

repeat()

[JavaScriptMethodStringRepeat]

The `repeat()` method (when applied to a JavaScript string) returns a string that repeats a specified string a specified number of times.

It has the following syntax with the following parameters:

string.`repeat(`*n*`)`

Parameter	Required / Optional	Description
n	Required	Number of times string is repeated

replace()

[JavaScriptMethodStringReplace]

The `replace()` method (when applied to a JavaScript string) returns a string that repeats a specified string a specified number of times.

It has the following syntax with the following parameters:

string.`replace(`*searchvalue*`)`

Parameter	Required / Optional	Description
searchvalue	Required	Value or regular expression to be replaced
newvalue	Required	New value inserted instead

If *searchvalue* is a normal string then only the first occurrence is replaced, if it occurs more than once in the string being searched. If you want to replace all occurrences then you need to use a corresponding regular expression with a `/g`, i.e. global, modifier.

search()

[JavaScriptMethodStringSearch]

The `search()` method (when applied to a JavaScript string) returns the position of the first occurrence of the search value (or -1, if no match is found).

It has the following syntax with the following parameters:

string.`search(`*regexpression*`)`

Parameter	Required / Optional	Description
regexpression	Required	Value (regular expression) to be searched for

slice()

[JavaScriptMethodStringSlice]

The `slice()` method (when applied to a JavaScript string) returns a new string formed by a part of the original string.

It has the following syntax with the following parameters:

string.`slice`(*n1*,*n2*)

Parameter	Required / Optional	Description
n1	Required	Position from where to begin extraction. `0` corresponds to the first character.
n2	Optional	(default is, in effect, *string*.`length`). Where to end extraction, so if omitted will select all characters from the *n1*'th position to the end of the string

split()

[JavaScriptMethodStringSplit]

The `split()` method (when applied to a JavaScript string) returns an array of substrings that are created by splitting the original string using a given delimiter.

It has the following syntax with the following parameters:

string.`split`(*delimiter*,*limit*)

Parameter	Required / Optional	Description
delimiter	Optional	Separator used to delimit individual entries. If the delimiter is `""` (i.e. an empty string) then the string is split between each character. If the delimiter is not present then split does not affect the original string
limit	Optional	An integer specifying maximum number of splits (items after the limit will not be included in output array)

startsWith()

[JavaScriptMethodStringStartsWith]

The `startsWith()` method (when applied to a JavaScript string) returns true if the string starts with a specified string, otherwise returns false.

It has the following syntax with the following parameters:

string.`startsWith`(*searchvalue*,*startposition*)

Parameter	Required / Optional	Description
searchvalue	Required	Value to be searched for
startposition	Optional	(default is 0). Position in underlying string from which to search from

substr()

[JavaScriptMethodStringSubstr]

The `substr()` method (when applied to a JavaScript string) returns a substring defined by the start position and number of characters.

It has the following syntax with the following parameters:

string.`substr(`*startposition*`,` *n*`)`

Parameter	Required / Optional	Description
startposition	Required	Position from where to start returned string. First character is at position 0. If *startposition* is positive and greater than or equal to *string*.`length` then returns an empty string. If *startposition* is negative then indicates number of characters before end from which to start (and if it is negative and larger in absolute value than the length of the string then a *startposition* of zero is used.
n	Optional	(default is *string*.`length`). Number of characters to return, if omitted returns whole of rest of string

substring()

[JavaScriptMethodStringSubstring]

The `substring()` method (when applied to a JavaScript string) returns a substring defined by the start position and number of characters.

It has the following syntax with the following parameters:

string.`substring(`*startposition*`,` *endposition*`)`

Parameter	Required / Optional	Description
startposition	Required	Position from where to start returned string. First character is at position 0.
endposition	Optional	(default is *string*.`length`). Position up to (but not including) where characters are returned from. If omitted then returns rest of string. If before *startposition* then the two are treated as if they were reversed.

toLocaleLowerCase()

[JavaScriptMethodStringToLocaleLowerCase]

The `toLocaleLowerCase()` method (when applied to a JavaScript string) returns a string that is the original string converted to lower case characters, bearing in mind the language settings of the browser (so sometimes does not return the same as `toLowerCase`).

It has the following syntax (with no parameters):

string.`toLocaleLowerCase()`

toLocaleUpperCase()

[JavaScriptMethodStringToLocaleUpperCase]

The `toLocaleUpperCase()` method (when applied to a JavaScript string) returns a string that is the original string converted to lower case characters, bearing in mind the language settings of the browser (so sometimes does not return the same as `toUpperCase`).

It has the following syntax (with no parameters):

string.`toLocaleUpperCase()`

toLowerCase()

[JavaScriptMethodStringToLowerCase]

The `toLowerCase()` method (when applied to a JavaScript string) returns a string that is the original string converted to lower case characters, bearing in mind the language settings of the browser.

It has the following syntax (with no parameters):

string.`toLowerCase()`

toString()

[JavaScriptMethodStringToString]

The `toString()` method (when applied to a JavaScript string) returns the string value of the string (i.e. itself).

It has the following syntax (with no parameters):

string.`toString()`

toUpperCase()

[JavaScriptMethodStringToUpperCase]

The `toUpperCase()` method (when applied to a JavaScript string) returns a string that is the original string converted to lower case characters, bearing in mind the language settings of the browser.

It has the following syntax (with no parameters):

string.`toUpperCase()`

trim()

[JavaScriptMethodStringTrim]

The `trim()` method (when applied to a JavaScript string) returns a string with whitespace (i.e. spaces) removed from start and finish of string.

It has the following syntax (with no parameters):

string.`trim()`

valueOf()

[JavaScriptMethodStringValueOf]

The `valueOf()` method (when applied to a JavaScript string) returns the string value of the string (i.e. itself).

It has the following syntax (with no parameters):

error.`valueOf()`

Appendix Q: JavaScript Regular Expressions

[JavaScriptTutorialRegularExpressions]

Some JavaScript string methods and properties involve 'regular expressions'. These take the form:

/pattern/modifiers

e.g.:

```
var x = /nematrian/i;
```

Modifiers can be `i`, `g` or `m` (or a combination). These have the following interpretations:

Modifier	Description	More
`g`	Do global match (i.e. find all matches rather than just first one)	Here
`i`	Do case-insensitive match	Here
`m`	Do a multiline match	Here

Regular expressions can include brackets to accept (or reject) a range of characters:

Expression	Description	More
`[xyz]`	Find any character within bracket	Here
`[^xyz]`	Find any character not within bracket	Here
`[0-9]`	Find any character within a range, here any digit	Here
`[^0-9]`	Find any character not within a range, here any digit	Here
`[a|b|cd]`	Find any from a set of specific alternatives	Here

Some characters appearing in regular expressions have special meanings:

Expression	Description	More
`.`	A single character, other than a new line or line terminator	Here
`\0`	A NUL character	Here
`\xxx`	The character specified by given octal number `xxx` (where each `x` is a decimal digit)	Here
`\b`	Match at beginning/end of word	Here
`\B`	Match not at beginning/end of word	Here
`\d`	A digit	Here
`\D`	A non-digit	Here
`\f`	A form feed	Here
`\n`	A new line	Here
`\r`	A carriage return	Here
`\s`	A whitespace	Here
`\S`	A non-whitespace	Here
`\t`	A tab	Here
`\uhhhh`	The character specified by (Unicode) hexadecimal number `hhhh` (where each `h` is a hexadecimal digit)	Here
`\v`	A vertical tab	Here
`\w`	A word character	Here
`\W`	A non-word character	Here

`\xhh`	Character specified by (Ascii) hexadecimal number `hh`, where each `h` is a hexadecimal digit)	Here

Regular expressions can also have quantifiers with the following meanings, where *s* is a specified string:

Quantifier	Description	More
s+	String that contains at least one *s*	Here
*s**	String that contains zero or more occurrences of *s*	Here
s?	String that contains zero or one occurrences of *s*	Here
s{*n*}	String that contains a sequence of *n s*'s	Here
s{*n1*,*n2*}	String that contains a sequence of *n1* to *n2 s*'s	Here
s{*n*,}	String that contains a sequence of at least *n s*'s	Here
s$	String that has *s* at the end	Here
^*s*	String that has *s* at the start	Here
?=*s*	String that is followed by *s*	Here
?!*s*	String that is not followed by *s*	Here

Regular expressions support the following properties and methods:

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`global`	Indicates if the "g" modifier is set	Here
`ignoreCase`	Indicates if the "i" modifier is set	Here
`lastIndex`	Indicates the index at which to start the next match	Here
`multiline`	Indicates if the "m" modifier is set	Here
`source`	Returns the text of the regular expression pattern	Here

Methods:

Method	Description	More
`exec()`	Seeks a match in a string and returns the first match	Here
`test()`	Seeks a match in a string and returns `true` if found, otherwise `false`	Here
`toString()`	Returns the (string) value of a regular expression	Here

The `compile()` method is depreciated and it is therefore recommended that it is not used.

Regular expression modifiers:

i modifier

[JavaScriptRegExprModifierI]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The `i` modifier indicates that JavaScript should do a case-insensitive match.

g modifier

[JavaScriptRegExprModifierG]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The `g` modifier indicates that JavaScript should do a global match (i.e. find all matches rather than just first one).

m modifier

[JavaScriptRegExprModifierM]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The `m` modifier indicates that JavaScript should do a multiline match.

Regular expression find pattern brackets:

[xyz]

[JavaScriptRegExprFind1]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

Regular expressions can also include brackets to accept (or reject) a range of characters. An expression containing `[xyz]` means find any character within the bracket.

[^xyz]

[JavaScriptRegExprFind2]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

Regular expressions can also include brackets to accept (or reject) a range of characters. An expression containing `[^xyz]` means find any character not within the bracket.

[0-9]

[JavaScriptRegExprFind3]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

Regular expressions can also include brackets to accept (or reject) a range of characters. An expression containing `[0-9]` means find any character within a given range, here any digit.

[^0-9]

[JavaScriptRegExprFind4]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

Regular expressions can also include brackets to accept (or reject) a range of characters. An expression containing `[^0-9]` means find any character not within a given range, here any digit.

[a|b|cd]

[JavaScriptRegExprFind5]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

Regular expressions can also include brackets to accept (or reject) a range of characters. An expression containing `[a|b|cd]` means find any from a set of specific alternatives.

Regular expression characters with special meanings:

"." character

[JavaScriptRegExprStop]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character "`.`" in a regular expression has a special meaning, namely a single character, other than a new line or line terminator.

"\0" character

[JavaScriptRegExpr0]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\0` in a regular expression has a special meaning, namely a NUL character.

"\xxx" character

[JavaScriptRegExprXxx]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\xxx` in a regular expression (where each `x` is an octal digit) has a special meaning, namely the character specified by a given octal number `xxx`.

"\b" character

[JavaScriptRegExprB]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\b` in a regular expression has a special meaning, namely to match at the beginning/end of a word.

"\B" character

[JavaScriptRegExprBupper]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\B` in a regular expression has a special meaning, namely to match not at the beginning/end of a word.

"\d" character

[JavaScriptRegExprD]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\d` in a regular expression has a special meaning, namely a digit.

"\D" character

[JavaScriptRegExprDupper]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\D` in a regular expression has a special meaning, namely a non-digit.

"\f" character

[JavaScriptRegExprF]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\f` in a regular expression has a special meaning, namely a form feed.

"\n" character

[JavaScriptRegExprN]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\n` in a regular expression has a special meaning, namely a new line.

"\r" character

[JavaScriptRegExprR]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\r` in a regular expression has a special meaning, namely a carriage return.

"\s" character

[JavaScriptRegExprS]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\s` in a regular expression has a special meaning, namely a whitespace.

"\S" character

[JavaScriptRegExprSupper]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\S` in a regular expression has a special meaning, namely a non-whitespace.

"\t" character

[JavaScriptRegExprT]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\t` in a regular expression has a special meaning, namely a (horizontal) tab.

"\uhhhh" character

[JavaScriptRegExprUhhhh]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\uhhhh` in a regular expression (where each `h` is a hexadecimal digit) has a special meaning, namely the character specified by the (Unicode) hexadecimal number `hhhh`.

"\v" character

[JavaScriptRegExprV]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\v` in a regular expression has a special meaning, namely a vertical tab.

"\w" character

[JavaScriptRegExprW]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\w` in a regular expression has a special meaning, namely a word character.

“\W” character

[JavaScriptRegExprWupper]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\W` in a regular expression has a special meaning, namely a non-word character.

“\xhh” character

[JavaScriptRegExprXhh]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

The character sequence `\xhh` in a regular expression (where each `h` is a hexadecimal digit) has a special meaning, namely the character specified by a given hexadecimal number `hh`.

Regular expression quantifiers:

“+” quantifier

[JavaScriptRegExprPlus]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression has a “+” qualifier, i.e. takes the form *s*+, then this means that it contains at least one *s*.

“*” quantifier

[JavaScriptRegExprStar]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression has a “*” qualifier, i.e. takes the form *s**, then this means that it contains zero or more occurrences of *s*.

“?” quantifier

[JavaScriptRegExprQuery]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression has a “?” qualifier, i.e. takes the form *s*?, then this means that it contains zero or one occurrences of *s*.

“*s{n}*” quantifier

[JavaScriptRegExprIndex1]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form *s*{*n*} then this means that it contains a sequence of *n s*’s.

“*s{n1,n2}*” quantifier

[JavaScriptRegExprIndex2]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form *s*{*n1*,*n2*} then this means that it contains a sequence of *n1* to *n2 s*’s.

“*s{n,}*” quantifier

[JavaScriptRegExprIndex3]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form *s*{*n*,} then this means that it contains a sequence of at least *n s*’s.

“*s$*” quantifier

[JavaScriptRegExprDollar]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form *s*$ then this means that it has *s* at the end.

“*^s*” quantifier

[JavaScriptRegExprUp]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form ^*s* then this means that it has *s* at the start.

“*?=s*” quantifier

[JavaScriptRegExprQueryEq]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form ?=*s* then this means that it is a string followed by *s*.

“?!*s*” quantifier

[JavaScriptRegExprQueryExclamation]

A JavaScript regular expression takes the form: /*pattern*/*modifiers*

If a component of a regular expression takes the form `?!`*s* then this means that it is a string not followed by *s*.

regular expression properties:

global

[JavaScriptPropertyRegExprGlobal]

The `global` property of a JavaScript regular expression indicates if the `g` modifier is set within the regular expression.

ignoreCase

[JavaScriptPropertyRegExprIgnoreCase]

The `ignoreCase` property of a JavaScript regular expression indicates if the `i` modifier is set within the regular expression.

lastIndex

[JavaScriptPropertyRegExprLastIndex]

The `lastIndex` property of a JavaScript regular expression indicates the index value at which to start the next match.

multiline

[JavaScriptPropertyRegExprMultiline]

The `multiline` property of a JavaScript regular expression indicates if the `m` modifier is set within the regular expression.

source

[JavaScriptPropertyRegExprSource]

The `source` property of a JavaScript regular expression returns the text of the regular expression pattern.

regular expression methods:

exec()

[JavaScriptMethodRegExprExec]

The `exec()` method (when applied to a JavaScript regular expression) seeks a match in a string and returns the first match.

It has the following syntax with the following parameters. It returns a string as above, or `null` if no such string is found.

RegExprObject.`exec(`*string*`)`

Parameter	Required / Optional	Description
string	Required	The string to be searched

test()

[JavaScriptMethodRegExprTest]

The `test()` method (when applied to a JavaScript regular expression) seeks a match in a string and returns `true` if found, otherwise `false`.

It has the following syntax with the following parameters. It returns a Boolean as above.

RegExprObject.`test(`*string*`)`

Parameter	Required / Optional	Description
string	Required	The string to be searched

toString()

[JavaScriptMethodRegExprToString]

The `toString()` method (when applied to a JavaScript regular expression) returns the string value of the regular expression.

It has the following syntax with no parameters. It returns a Boolean as above.

RegExprObject.`toString()`

Appendix R: JavaScript Numbers and Mathematical Functions

[JavaScriptTutorialNumbers]

JavaScript has only one type of number (in contrast to, e.g. Visual Basic, which differentiates between e.g. integers, floating point numbers and double precision numbers). Numbers can be written with or without decimal points and/or with or without (scientific) exponents, e.g.

```
var x = 4.1;     // With a decimal point
var y = 4;       // Without a decimal point
var p = 135e6    // Means 135000000
var q = 13.5e-3 // Means 0.0135
```

Numbers have the following properties and methods:

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`MAX_VALUE`	Returns largest (positive) number recognised by the browser's JavaScript	Here
`MIN_VALUE`	Returns smallest (positive) number recognised by the browser's JavaScript	Here
`NEGATIVE_INFINITY`	Represents negative infinity (i.e. if computation overflows)	Here
`NaN`	Represents Not-a-Number (i.e. if computation overflows	Here
`POSITIVE_INFINITY`	Represents positive infinity (i.e. if computation overflows)	Here
`prototype`	Allows author to add properties and methods to an object	Here

Methods:

Method	Description	More
`isFinite()`	Returns true if value is a finite number, otherwise returns false	Here
`isInteger()`	Returns true if value is of type Number and is an integer (within range understood as integers by the browser), otherwise returns false	Here
`isNaN()`	Returns true if value is NaN, otherwise returns false	Here
`isSafeInteger()`	Returns true if value is of type Number and is a safe integer, otherwise false. A safe integer is one that can be exactly represented as an IEEE-754 double precision number, i.e. is an integer in the range $-(2^{53} - 1)$ to $(2^{53} - 1)$.	Here
`toExponential()`	Returns a string representing the number converted into exponential form. The optional parameter (0 to 20) represents the number of digits retained after the decimal point	Here
`toFixed()`	Returns a string representing the number with a	Here

	fixed number of digits after the decimal point	
`toPrecision()`	Returns a string representing the number with a fixed number of significant digits	Here
`toString()`	Returns a string corresponding to the number	Here
`valueOf()`	Returns the primitive value of an object. For a number, this in effect just returns the number itself	Here

The Math object

Associated with numbers is the JavaScript Math object. This allows authors to carry out some mathematical manipulations. It supports the following properties and methods:

Math object properties:

Property	Description	More
`E`	Returns Euler's constant, e	Here
`LN2`	Returns the natural logarithm of `2`	Here
`LN10`	Returns the natural logarithm of `10`	Here
`LOG2E`	Returns the base-2 logarithm of e	Here
`LOG10E`	Returns the base-10 logarithm of e	Here
`PI`	Returns π	Here
`SQRT1_2`	Returns $1/\sqrt{2}$	Here
`SQRT2`	Returns $\sqrt{2}$	Here

Math object methods:

Method	Description	More
`abs()`	Returns the absolute value of a real number	Here
`acos()`	Returns the (principal) arccosine of a real number	Here
`acosh()`	Returns the (principal) hyperbolic arccosine of a real number	Here
`asin()`	Returns the (principal) arcsine of a real number	Here
`asinh()`	Returns the (principal) hyperbolic arcsine of a real number	Here
`atan()`	Returns the (principal) arctangent of a real number	Here
`atanh()`	Returns the (principal) hyperbolic arctangent of a real number	Here
`atan2()`	Returns the arctangent of the specified x- and y-coordinates	Here
`cbrt()`	Returns the cube root of a real number	Here
`ceil()`	Rounds a real number towards positive infinity	Here
`cos()`	Returns the cosine of a real number	Here
`cosh()`	Returns the hyperbolic cosine of a real number	Here
`exp()`	Returns the exponential of a real number (i.e. e^x)	Here
`floor()`	Rounds a real number towards negative infinity	Here
`log()`	Returns the natural logarithm of a positive real number	Here
`max()`	Returns the maximum of a set of real numbers	Here
`min()`	Returns the minimum of a set of real numbers	Here
`pow()`	Returns *x* to the power *y*. Note, ^ has a different	Here

	meaning in JavaScript	
`random()`	Returns a (uniform) random number between 0 and 1	Here
`round()`	Rounds a real number to the nearest integer	Here
`sin()`	Returns the sine of a real number	Here
`sinh()`	Returns the hyperbolic sine of a real number	Here
`sqrt()`	Returns the square root of a real (non-negative) number	Here
`tan()`	Returns the tangent of a real number	Here
`tan()`	Returns the hyperbolic tangent of a real number	Here

Number properties:

MAX_VALUE

[JavaScriptPropertyNumberMaxValue]

The `MAX_VALUE` property (of the JavaScript Number object) returns the largest finite value acceptable in JavaScript.

It has the following syntax:

```
Number.MAX_VALUE
```

MIN_VALUE

[JavaScriptPropertyNumberMinValue]

The `MIN_VALUE` property (of the JavaScript Number object) returns the smallest (positive) value acceptable in JavaScript.

It has the following syntax:

```
Number.MIN_VALUE
```

NaN

[JavaScriptPropertyNumberNaN]

The `NaN` property (of the JavaScript Number object) returns `NaN` (i.e. 'not a number').

It has the following syntax:

```
Number.NaN
```

NEGATIVE_INFINITY

[JavaScriptPropertyNumberNegativeInfinity]

The `NEGATIVE_INFINITY` property (of the JavaScript Number object) returns negative infinity.

It has the following syntax:

```
Number.NEGATIVE_INFINITY
```

POSITIVE_INFINITY

[JavaScriptPropertyNumberPositiveInfinity]

The `POSITIVE_INFINITY` property (of the JavaScript Number object) returns positive infinity.

It has the following syntax:

```
Number.POSITIVE_INFINITY
```

Number methods:

isFinite()

[JavaScriptMethodNumberIsFinite]

The `isFinite()` method (of the JavaScript Number object) returns `true` if value is a finite number, otherwise returns `false`.

It has the following syntax with the following parameters:

```
Number.isFinite(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

The `Number.isFinite` method is subtly different to the global isFinite function. The latter coerces a value to a number before testing it, whilst the former does not. So, `Number.isFinite("4.3")` returns `false`, whilst `isFinite("4.3")` returns `true`.

isInteger()

[JavaScriptMethodNumberIsInteger]

The `isInteger()` method (of the JavaScript Number object) returns true if value is of type Number and is an integer (within range understood as integers by the browser), otherwise returns false.

It has the following syntax with the following parameters:

```
Number.isInteger(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

isNaN()

[JavaScriptMethodNumberIsNaN]

The `isNaN()` method (of the JavaScript Number object) returns `true` if value is of type Number and is an integer (within range understood as integers by the browser), otherwise returns `false`.

It has the following syntax with the following parameters:

```
Number.isNaN(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

The `Number.isNaN` method is subtly different to the global isNaN function. The latter coerces a value to a number before testing it, whilst the former does not. So, `Number.isNaN("NaN")` returns `false`, whilst `isNaN("NaN")` returns `true`.

isSafeInteger()

[JavaScriptMethodNumberIsSafeInteger]

The `isSafeInteger()` method (of the JavaScript Number object) returns true if value is of type Number and is a safe integer, otherwise false. A safe integer is one that can be exactly represented as an IEEE-754 double precision number, i.e. is an integer in the range -(2^{53} - 1) to (2^{53} - 1).

It has the following syntax with the following parameters:

```
Number.isSafeInteger(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

toExponential()

[JavaScriptMethodNumberToExponential]

The `toExponential()` method (when applied to JavaScript numbers) returns a string representing the number in exponential notation, e.g. 301 is 3.01e+2.

It has the following syntax with the following parameters:

```
number.toExponential(n)
```

Parameter	Required / Optional	Description
n	Optional	Integer between 0 and 20 indicating number of digits after decimal point

toFixed()

[JavaScriptMethodNumberToFixed]

The `toFixed()` method (when applied to JavaScript numbers) returns a string representing the number with a fixed number of digits after the decimal point.

It has the following syntax with the following parameters:

number`.toFixed(`*n*`)`

Parameter	Required / Optional	Description
n	Optional	(default is `0`), Integer indicating number of digits after decimal point

toPrecision()

[JavaScriptMethodNumberToPrecision]

The `toPrecision()` method (when applied to JavaScript numbers) returns a string representing the number with a fixed number of significant digits.

It has the following syntax with the following parameters:

number`.toPrecision(`*n*`)`

Parameter	Required / Optional	Description
n	Optional	(default is `0`), Integer indicating number of digits. If omitted then returns a string representation of the entire number (without any formatting)

toString()

[JavaScriptMethodNumberToString]

The `toString()` method (when applied to JavaScript numbers) returns a string corresponding to the number.

It has the following syntax (with no parameters):

number`.toString()`

valueOf()

[JavaScriptMethodNumberValueOf]

The `valueOf()` method (when applied to JavaScript numbers) returns the primitive value of the number (i.e. itself).

It has the following syntax (with no parameters):

number`.valueOf()`

Math Object properties

E

[JavaScriptPropertyMathE]

The `E` property (of the Math object) returns (Euler's) constant, e, i.e. the limit of $(1 + 1/n)^n$ as n tends to plus infinity.

It has the following syntax:

```
Math.E
```

LN2

[JavaScriptPropertyMathLN2]

The `LN2` property (of the Math object) returns the natural logarithm of `2`.

It has the following syntax:

```
Math.LN2
```

LN10

[JavaScriptPropertyMathLN10]

The `LN10` property (of the Math object) returns the natural logarithm of `10`.

It has the following syntax:

```
Math.LN10
```

LOG2E

[JavaScriptPropertyMathLOG2E]

The `LOG2E` property (of the Math object) returns the base-2 logarithm of e.

It has the following syntax:

```
Math.LOG2E
```

LOG10E

[JavaScriptPropertyMathLOG10E]

The `LOG10E` property (of the Math object) returns the base-10 logarithm of e.

It has the following syntax:

```
Math.LOG10E
```

PI

[JavaScriptPropertyMathPi]

The `PI` property (of the Math object) returns π (the ratio of a circle's circumference to its diameter).

It has the following syntax:

```
Math.PI
```

SQRT1_2

[JavaScriptPropertyMathSqrt1over2]

The `SQRT1_2` property (of the Math object) returns $1/\sqrt{2}$.

It has the following syntax:

```
Math.SQRT1_2
```

SQRT2

[JavaScriptPropertyMathSqrt2]

The `SQRT2` property (of the Math object) returns $\sqrt{2}$.

It has the following syntax:

```
Math.SQRT2
```

Math Object methods:

abs()

[JavaScriptMethodMathAbs]

The `abs()` method (of the Math object) returns the absolute value of a real number.

It has the following syntax with the following parameters:

```
Math.abs(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

acos()

[JavaScriptMethodMathAcos]

The `acos()` method (of the Math object) returns the (principal) arccosine of a real number.

It has the following syntax with the following parameters:

```
Math.acos(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

acosh()

[JavaScriptMethodMathAcosh]

The `acosh()` method (of the Math object) returns the (principal) hyperbolic arccosine of a real number.

It has the following syntax with the following parameters:

```
Math.acosh(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

asin()

[JavaScriptMethodMathAsin]

The `asin()` method (of the Math object) returns the (principal) arcsine of a real number.

It has the following syntax with the following parameters:

```
Math.asin(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

asinh()

[JavaScriptMethodMathAsinh]

The `asinh()` method (of the Math object) returns the (principal) hyperbolic arcsine of a real number.

It has the following syntax with the following parameters:

```
Math.asinh(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

atan()

[JavaScriptMethodMathAtan]

The `atan()` method (of the Math object) returns the (principal) arctangent of a real number.

It has the following syntax with the following parameters:

```
Math.atan(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

atanh()

[JavaScriptMethodMathAtanh]

The `atanh()` method (of the Math object) returns the (principal) hyperbolic arctangent of a real number.

It has the following syntax with the following parameters:

```
Math.atanh(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

atan2()

[JavaScriptMethodMathAtan2]

The `atan2()` method (of the Math object) returns the (principal) arctangent of a real number.

It has the following syntax with the following parameters:

```
Math.atan2(y,x)
```

Parameter	Required / Optional	Description
y	Required	*y*-coordinate
x	Required	*x*-coordinate

Note: many computer languages have an atan2 function, but the ordering of the parameters is not the same across all languages

cbrt()

[JavaScriptMethodMathCbrt]

The `cbrt()` method (of the Math object) returns the cube root of a real number.

It has the following syntax with the following parameters:

```
Math.cbrt(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

ceil()

[JavaScriptMethodMathCeil]

The `ceil()` method (of the Math object) rounds a real number towards positive infinity.

It has the following syntax with the following parameters:

```
Math.ceil(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

cos()

[JavaScriptMethodMathCos]

The `cos()` method (of the Math object) returns the cosine of a real number.

It has the following syntax with the following parameters:

```
Math.cos(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

cosh()

[JavaScriptMethodMathCosh]

The `cosh()` method (of the Math object) returns the hyperbolic cosine of a real number.

It has the following syntax with the following parameters:

```
Math.cosh(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

exp()

[JavaScriptMethodMathExp]

The `exp()` method (of the Math object) returns the exponential of a real number (i.e. e^x).

It has the following syntax with the following parameters:

`Math.exp(`*x*`)`

Parameter	Required / Optional	Description
x	Required	Input parameter

floor()

[JavaScriptMethodMathFloor]

The `floor()` method (of the Math object) rounds a real number towards negative infinity.

It has the following syntax with the following parameters:

`Math.floor(`*x*`)`

Parameter	Required / Optional	Description
x	Required	Input parameter

log()

[JavaScriptMethodMathLog]

The `log()` method (of the Math object) returns the natural logarithm of a positive real number.

It has the following syntax with the following parameters:

`Math.log(`*x*`)`

Parameter	Required / Optional	Description
x	Required	Input parameter

max()

[JavaScriptMethodMathMax]

The `max()` method (of the Math object) returns the maximum of a set of real numbers.

It has the following syntax with the following parameters:

`Math.max(`*x1, x2, x3, …*`)`

Parameter	Required / Optional	Description
x1, x2, x3, …	Required	Input values

You can find the maximum of an array using a format such as:

`Math.max.apply(null,` *xarray*`)`

since e.g. `Math.max.apply(null, [1, 2, 3])` is equivalent to `Math.max(1, 2, 3)`

min()

[JavaScriptMethodMathMin]

The `min()` method (of the Math object) returns the minimum of a set of real numbers.

It has the following syntax with the following parameters:

`Math.min(`*x1, x2, x3, …*`)`

Parameter	Required / Optional	Description
x1, x2, x3, …	Required	Input values

You can find the minimum of an array using a format such as:

`Math.min.apply(null,` *xarray*`)`

since e.g. `Math.min.apply(null, [1, 2, 3])` is equivalent to `Math.min(1, 2, 3)`

pow()

[JavaScriptMethodMathPow]

The `pow()` method (of the Math object) returns *x* to the power *y*. Note, ^ has a different meaning in JavaScript.

It has the following syntax with the following parameters:

`Math.pow(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	Input value (the base)
y	Required	Input value (the exponent)

random()

[JavaScriptMethodMathRandom]

The `random()` method (of the Math object) returns a (uniform) random number between 0 (inclusive) and 1 (not inclusive).

It has the following syntax (with no parameters):

```
Math.random()
```

round()

[JavaScriptMethodMathRound]

The `round()` method (of the Math object) rounds a real number to the nearest integer.

It has the following syntax with the following parameters:

```
Math.round(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

sin()

[JavaScriptMethodMathSin]

The `sin()` method (of the Math object) returns the sine of a real number.

It has the following syntax with the following parameters:

```
Math.sin(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

sinh()

[JavaScriptMethodMathSinh]

The `sinh()` method (of the Math object) returns the hyperbolic sine of a real number.

It has the following syntax with the following parameters:

```
Math.sinh(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

sqrt()

[JavaScriptMethodMathSqrt]

The `sqrt()` method (of the Math object) returns the square root of a real (non-negative) number.

It has the following syntax with the following parameters:

```
Math.sqrt(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

tan()

[JavaScriptMethodMathTan]

The `tan()` method (of the Math object) returns the tangent of a real number.

It has the following syntax with the following parameters:

```
Math.tan(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

tanh()

[JavaScriptMethodMathTanh]

The `tanh()` method (of the Math object) returns the hyperbolic tangent of a real number.

It has the following syntax with the following parameters:

```
Math.tanh(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

Appendix S: JavaScript Dates

[JavaScriptTutorialDates]

JavaScript date variables are objects and contain dates and times. They can be instantiated in 4 ways:

```
var d1 = new Date();            // An as yet undefined date
var d2 = new Date(milliseconds);   // See below
var d3 = new Date(dateString);    // See below
var d4 = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

Here *milliseconds* refers to the number of milliseconds since 1 January 1970 00:00:00. A *dateString* is a textual representation of a date.

The Date object supports the following properties and methods:

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`prototype`	Allows author to add properties and methods to an object	Here

Methods:

Method	Description	More
`getDate()`	Returns day of month (1 to 31)	Here
`getDay()`	Returns day of week (0 to 6)	Here
`getFullYear()`	Returns year	Here
`getHours()`	Returns hour (0 to 23)	Here
`getMilliseconds()`	Returns milliseconds (0 to 999)	Here
`getMinutes()`	Returns minutes (0 to 59)	Here
`getMonth()`	Returns month (0 to 11)	Here
`getSeconds()`	Returns seconds (0 to 59)	Here
`getTime()`	Returns number of milliseconds since 1 January 1970 00:00:00	Here
`getTimezoneOffset()`	Returns time difference between UTC time and local time, in minutes	Here
`getUTCDate()`	Returns UTC day of month (1 to 31)	Here
`getUTCDay()`	Returns UTC day of week (0 to 6)	Here
`getUTCFullYear()`	Returns UTC year	Here
`getUTCHours()`	Returns UTC hour (0 to 23)	Here
`getUTCMilliseconds()`	Returns UTC milliseconds (0 to 999)	Here
`getUTCMinutes()`	Rounds UTC minutes (0 to 59)	Here
`getUTCMonth()`	Returns UTC month (0 to 11)	Here
`getUTCSeconds()`	Returns UTC seconds (0 to 59)	Here
`getYear()`	*Depreciated*. Use `getFullYear()` instead	Here
`now()`	Returns current date and time, as number of milliseconds since 1 January 1970 00:00:00	Here
`parse()`	Parses a *dateString* and returns the number of	Here

	milliseconds since 1 January 1970 00:00:00	
`setDate()`	Sets day of month	Here
`setFullYear()`	Sets year (and optionally month and day)	Here
`setHours()`	Sets hours (and optionally minutes, seconds and milliseconds)	Here
`setMilliseconds()`	Sets miliseconds	Here
`setMinutes()`	Sets minutes (and optionally seconds and milliseconds)	Here
`setMonth()`	Sets month (and optionally day)	Here
`setSeconds()`	Sets seconds (and optionally milliseconds)	Here
`setTime()`	Sets a date given a specified number of milliseconds since 1 January 1970 00:00:00	Here
`setUTCDate()`	Sets UTC day of month	Here
`setUTCFullYear()`	Sets UTC year (and optionally month and day)	Here
`setUTCHours()`	Sets UTC hours (and optionally minutes, seconds and milliseconds)	Here
`setUTCMilliseconds()`	Sets UTC miliseconds	Here
`setUTCMinutes()`	Sets UTC minutes (and optionally seconds and milliseconds)	Here
`setUTCMonth()`	Sets UTC month (and optionally day)	Here
`setUTCSeconds()`	Sets UTC seconds (and optionally milliseconds)	Here
`setYear()`	*Depreciated*. Use `setFullYear()` instead	
`toDateString()`	Returns date portion as a string	Here
`toGMTString()`	*Depreciated*. Use `toUTCString()` instead	
`toISOString()`	Returns date as a string, using ISO notation	Here
`toJSON()`	Returns date as a string, using JSON notation	Here
`toLocaleDateString()`	Returns date portion as a string, using locale-specified notation	Here
`toLocaleTimeString()`	Returns time portion as a string, using locale-specified notation	Here
`toLocaleString()`	Returns date (and time) as a string, using locale-specified notation	Here
`toString()`	Returns date (and time) as a string	Here
`toUTCString()`	Returns UTC date (and time) as a string	Here
`UTC()`	Returns number of UTC milliseconds since 1 January 1970 00:00:00	Here
`valueOf()`	Returns the primitive value of the object	Here

Date methods

getDate()

[JavaScriptMethodDateGetDate]

The `getDate()` method (when applied to a JavaScript date) returns the day of the month (1 to 31).

It has the following syntax (with no parameters):

date`.getDate()`

getDay()

[JavaScriptMethodDateGetDay]

The `getDay()` method (when applied to a JavaScript date) returns the day of the week (0 to 6).

It has the following syntax (with no parameters):

date`.getDay()`

getFullYear()

[JavaScriptMethodDateGetFullYear]

The `getFullYear()` method (when applied to a JavaScript date) returns the year.

It has the following syntax (with no parameters):

date`.getFullYear()`

getHours()

[JavaScriptMethodDateGetHours]

The `getHours()` method (when applied to a JavaScript date) returns the hour (0 to 23).

It has the following syntax (with no parameters):

date`.getHours()`

getMilliseconds()

[JavaScriptMethodDateGetMilliseconds]

The `getMilliseconds()` method (when applied to a JavaScript date) returns the milliseconds (0 to 999).

It has the following syntax (with no parameters):

date`.getMilliseconds()`

getMinutes()

[JavaScriptMethodDateGetMinutes]

The `getMinutes()` method (when applied to a JavaScript date) returns the minutes (0 to 59).

It has the following syntax (with no parameters):

date`.getMinutes()`

getMonth()

[JavaScriptMethodDateGetMonth]

The `getMonth()` method (when applied to a JavaScript date) returns the month (0 to 11).

It has the following syntax (with no parameters):

date`.getMonth()`

getSeconds()

[JavaScriptMethodDateGetSeconds]

The `getSeconds()` method (when applied to a JavaScript date) returns the seconds (0 to 59).

It has the following syntax (with no parameters):

date`.getSeconds()`

getTime()

[JavaScriptMethodDateGetTime]

The `getTime()` method (when applied to a JavaScript date) returns the number of milliseconds since 1 January 1970 00:00:00.

It has the following syntax (with no parameters):

date`.getTime()`

getTimezoneOffset()

[JavaScriptMethodDateGetTimezoneOffset]

The `getTimezoneOffset()` method (when applied to a JavaScript date) returns the time difference between UTC time and local time, in minutes.

It has the following syntax (with no parameters):

date`.getTimezoneOffset()`

getUTCDate()

[JavaScriptMethodDateGetUTCDate]

The `getUTCDate()` method (when applied to a JavaScript date) returns the UTC day of the month (1 to 31).

It has the following syntax (with no parameters):

date`.getUTCDate()`

getUTCDay()

[JavaScriptMethodDateGetUTCDay]

The `getUTCDay()` method (when applied to a JavaScript date) returns the UTC day of the week (0 to 6).

It has the following syntax (with no parameters):

date`.getUTCDay()`

getUTCFullYears()

[JavaScriptMethodDateGetUTCFullYear]

The `getUTCFullYears()` method (when applied to a JavaScript date) returns the UTC year.

It has the following syntax (with no parameters):

date`.getUTCFullYears()`

getUTCHours()

[JavaScriptMethodDateGetUTCHours]

The `getUTCHours()` method (when applied to a JavaScript date) returns the UTC hour (0 to 23).

It has the following syntax (with no parameters):

date`.getUTCHours()`

getUTCMilliseconds()

[JavaScriptMethodDateGetUTCMilliseconds]

The `getUTCMilliseconds()` method (when applied to a JavaScript date) returns the UTC milliseconds (0 to 999).

It has the following syntax (with no parameters):

date`.getUTCMilliseconds()`

getUTCMinutes()

[JavaScriptMethodDateGetUTCMinutes]

The `getUTCMinutes()` method (when applied to a JavaScript date) returns the UTC minutes (0 to 59).

It has the following syntax (with no parameters):

date`.getUTCMinutes()`

getUTCMonth()

[JavaScriptMethodDateGetUTCMonth]

The `getUTCMonth()` method (when applied to a JavaScript date) returns the UTC month (0 to 11).

It has the following syntax (with no parameters):

date`.getUTCMonth()`

getUTCSeconds()

[JavaScriptMethodDateGetUTCSeconds]

The `getUTCSeconds()` method (when applied to a JavaScript date) returns the UTC seconds (0 to 59).

It has the following syntax (with no parameters):

date`.getUTCSeconds()`

getYear()

[JavaScriptMethodDateGetYear]

Depreciated. Use `getFullYear()` instead.

It has the following syntax (with no parameters):

date`.getYear()`

now()

[JavaScriptMethodDateNow]

The `now()` method (when applied to the JavaScript Date object) returns the current date and time, as number of milliseconds since 1 January 1970 00:00:00.

It has the following syntax (with no parameters):

`Date.now()`

parse()

[JavaScriptMethodDateParse]

The `parse()` method (when applied to the JavaScript Date object) Parses a *dateString* and returns the number of milliseconds since 1 January 1970 00:00:00.

It has the following syntax with the following parameters:

`Date.parse(`*dateString*`)`

Parameter	Required / Optional	Description
dateString	Required	A string representation of a date

setDate()

[JavaScriptMethodDateSetDate]

The `setDate()` method (when applied to a JavaScript date) sets the date variable's day of month.

It has the following syntax with the following parameters:

date`.setDate(`*day*`)`

Parameter	Required / Optional	Description
day	Required	Integer representing day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

setFullYear()

[JavaScriptMethodDateSetFullYear]

The `setFullYear()` method (when applied to a JavaScript date) sets the date variable's year (and optionally its month and day).

It has the following syntax with the following parameters:

date`.setFullYear(`*year*`,`*month*`,`*day*`)`

Parameter	Required / Optional	Description
year	Required	Integer (with 4 digits for years between 1000 and 9999).
month	Optional	Integer representing month of year. Typically, will be in range 0 – 11. However, -1 will result last month of previous year, 12 will result in first month of next year etc.
day	Optional	Integer representing day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of

		previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

setHours()

[JavaScriptMethodDateSetHours]

The `setHours()` method (when applied to a JavaScript date) sets the date variable's hour (and optionally its minute, second and millisecond).

It has the following syntax with the following parameters:

date`.setHours(`*hour*`,` *minute*`,` *second*`,` *millisecond*`)`

Parameter	Required / Optional	Description
hour	Required	Integer representing hour. Typically, will be in range 0 – 23. However, e.g. -1 will result in the last hour of the previous day, 24 will result in the first hour of the next day, etc.
minute	Optional	Integer representing minutes. Typically, will be in range 0 – 59. However, e.g. -1 will result in last minute of previous hour, 60 will result in first minute of next hour, etc.
second	Optional	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setMilliseconds()

[JavaScriptMethodDateSetMilliseconds]

The `setMilliseconds()` method (when applied to a JavaScript date) sets the date variable's millisecond.

It has the following syntax with the following parameters:

date`.setMilliseconds(`*millisecond*`)`

Parameter	Required / Optional	Description
millisecond	Required	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setMinutes()

[JavaScriptMethodDateSetMinutes]

The `setMinutes()` method (when applied to a JavaScript date) sets the date variable's minute (and optionally its second and millisecond).

It has the following syntax with the following parameters:

date`.setMinutes(`*minute*`,`*second*`,`*millisecond*`)`

Parameter	Required / Optional	Description
minute	Required	Integer representing minutes. Typically, will be in range 0 – 59. However, e.g. -1 will result in last minute of previous hour, 60 will result in first minute of next hour, etc.
second	Optional	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setMonth()

[JavaScriptMethodDateSetMonth]

The `setMonth()` method (when applied to a JavaScript date) sets the date variable's month (and optionally its day)

It has the following syntax with the following parameters:

date`.setMonth(`*month, day*`)`

Parameter	Required / Optional	Description
month	Required	Integer representing month of year. Typically, will be in range 0 – 11. However, -1 will result last month of previous year, 12 will result in first month of next year etc.
day	Optional	Integer representing day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

setSeconds()

[JavaScriptMethodDateSetSeconds]

The `setSeconds()` method (when applied to a JavaScript date) sets the date variable's second (and optionally its millisecond).

It has the following syntax with the following parameters:

date`.setSeconds(`*second*`,` *millisecond*`)`

Parameter	Required / Optional	Description
second	Required	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setTime()

[JavaScriptMethodDateSetTime]

The `setTime()` method (when applied to a JavaScript date) sets the date given a specified number of milliseconds since 1 January 1970 00:00:00.

It has the following syntax with the following parameters:

date`.setTime(`*second*`,` *millisecond*`)`

Parameter	Required / Optional	Description
millisecond	Required	Integer representing milliseconds since 1 January 1970 00:00:00

setUTCDate()

[JavaScriptMethodDateSetUTCDate]

The `setUTCDate()` method (when applied to a JavaScript date) sets the date variable's UTC day of month.

It has the following syntax with the following parameters:

date`.setUTCDate(`*day*`)`

Parameter	Required / Optional	Description
day	Required	Integer representing UTC day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

setUTCFullYear()

[JavaScriptMethodDateSetUTCFullYear]

The `setUTCFullYear()` method (when applied a JavaScript date) sets the date variable's UTC year (and optionally its month and day).

It has the following syntax with the following parameters:

date`.setUTCFullYear(`*year*`,`*month*`,`*day*`)`

Parameter	Required / Optional	Description
year	Required	Integer (with 4 digits for years between 1000 and 9999).
month	Optional	Integer representing month of year. Typically, will be in range 0 – 11. However, -1 will result last month of previous year, 12 will result in first month of next year etc.
day	Optional	Integer representing day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

setUTCHours()

[JavaScriptMethodDateSetUTCHours]

The `setUTCHours()` method (when applied to a JavaScript date) sets the date variable's UTC hour (and optionally its minute, second and millisecond).

It has the following syntax with the following parameters:

date`.setUTCHours(`*hour*`,`*minute*`,`*second*`,`*millisecond*`)`

Parameter	Required / Optional	Description
hour	Required	Integer representing hour. Typically, will be in range 0 – 23. However, e.g. -1 will result in the last hour of the previous day, 24 will result in the first hour of the next day, etc.
minute	Optional	Integer representing minutes. Typically, will be in range 0 – 59. However, e.g. -1 will result in last minute of previous hour, 60 will result in first minute of next hour, etc.
second	Optional	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in

		range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setUTCMilliseconds()

[JavaScriptMethodDateSetUTCMilliseconds]

The `setUTCMilliseconds()` method (when applied to a JavaScript date) sets the date variable's UTC millisecond.

It has the following syntax with the following parameters:

date`.setUTCMilliseconds(`*millisecond*`)`

Parameter	Required / Optional	Description
millisecond	Required	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setUTCMinutes()

[JavaScriptMethodDateSetUTCMinutes]

The `setUTCMinutes()` method (when applied to a JavaScript date) sets the date variable's UTC minute (and optionally its second and millisecond).

It has the following syntax with the following parameters:

date`.setUTCMinutes(`*minute*`,`*second*`,`*millisecond*`)`

Parameter	Required / Optional	Description
minute	Required	Integer representing minutes. Typically, will be in range 0 – 59. However, e.g. -1 will result in last minute of previous hour, 60 will result in first minute of next hour, etc.
second	Optional	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

setUTCMonth()

[JavaScriptMethodDateSetUTCMonth]

The `setUTCMonth()` method (when applied to a JavaScript date) sets the date variable's UTC month (and optionally its day)

It has the following syntax with the following parameters:

date`.setUTCMonth(`*month, day*`)`

Parameter	Required / Optional	Description
month	Required	Integer representing month of year. Typically, will be in range 0 – 11. However, -1 will result last month of previous year, 12 will result in first month of next year etc.
day	Optional	Integer representing day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

setUTCSeconds()

[JavaScriptMethodDateSetUTCSeconds]

The `setUTCSeconds()` method (when applied to a JavaScript date) sets the date variable's UTC second (and optionally its millisecond).

It has the following syntax with the following parameters:

date`.setUTCSeconds(`*second*`,` *millisecond*`)`

Parameter	Required / Optional	Description
second	Required	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

toDateString()

[JavaScriptMethodDateToDateString]

The `toDateString()` method (when applied to a JavaScript date) returns the date portion as a string.

It has the following syntax (with no parameters):

date`.toDateString()`

toISOString()

[JavaScriptMethodDateToISOString]

The `toISOString()` method (when applied to a JavaScript date) returns the date as a string, using ISO notation.

It has the following syntax (with no parameters):

date`.toISOString()`

toJSON()

[JavaScriptMethodDateToJSON]

The `toJSON()` method (when applied to a JavaScript date) returns the date as a string, using JSON notation.

It has the following syntax (with no parameters):

date`.toJSON()`

toLocaleDateString()

[JavaScriptMethodDateToLocaleDateString]

The `toLocaleDateString()` method (when applied to a JavaScript date) returns the date portion as a string, using locale-specified notation.

It has the following syntax (with no parameters):

date`.toLocaleDateString()`

toLocaleString()

[JavaScriptMethodDateToLocaleString]

The `toLocaleString()` method (when applied to a JavaScript date) returns the date (and time) as a string, using locale-specified notation.

It has the following syntax (with no parameters):

date`.toLocaleString()`

toLocaleTimeString()

[JavaScriptMethodDateToLocaleTimeString]

The `toLocaleTimeString()` method (when applied to a JavaScript date) returns the time portion as a string, using locale-specified notation.

It has the following syntax (with no parameters):

date`.toLocaleTimeString()`

toString()

[JavaScriptMethodDateToString]

The `toString()` method (when applied to a JavaScript date) returns the date (and time) as a string.

It has the following syntax (with no parameters):

date`.toString()`

toUTCString()

[JavaScriptMethodDateToUTCString]

The `toUTCString()` method (when applied to a JavaScript date) returns the UTC date (and time) as a string.

It has the following syntax (with no parameters):

date`.toUTCString()`

UTC()

[JavaScriptMethodDateUTC]

The `UTC()` method (when applied to the JavaScript Date object) returns number of UTC milliseconds since 1 January 1970 00:00:00.

It has the following syntax with the following parameters:

`Date.UTC(`*year*`,`*month*`,`*day*`,`*hour*`,`*minute*`,`*second*`,`*millisecond*`)`

Parameter	Required / Optional	Description
year	Required	Integer (with 4 digits for years between 1000 and 9999).
month	Required	Integer representing month of year. Typically, will be in range 0 – 11. However, -1 will result last month of previous year, 12 will result in first month of next year etc.
day	Optional	Integer representing day of month. Typically, will be in range 1 – 31. However, 0 will result in last day of previous month, -1 the day before that etc., and e.g. 32 for a 30-day month will be second day of following month

hour	Optional	Integer representing hour. Typically, will be in range 0 – 23. However, e.g. -1 will result in the last hour of the previous day, 24 will result in the first hour of the next day, etc.
minute	Optional	Integer representing minutes. Typically, will be in range 0 – 59. However, e.g. -1 will result in last minute of previous hour, 60 will result in first minute of next hour, etc.
second	Optional	Integer representing seconds. Typically, will be in range 0 – 59. However, e.g. 0 will result in last second of previous minute, 60 will result in first second of next minute, etc.
millisecond	Optional	Integer representing milliseconds. Typically, will be in range 0 – 999. However, e.g. 0 will result in last millisecond of previous second, 1000 will result in first millisecond of next second, etc.

valueOf()

[JavaScriptMethodDateValueOf]

The `valueOf()` method (when applied to a JavaScript date) returns the primitive value of the date.

It has the following syntax (with no parameters):

date`.valueOf()`

Appendix T: JavaScript Booleans

[JavaScriptTutorialBooleans]

JavaScript Boolean variables take one of two values, `true` or `false`. They are instantiated by a statement such as:

```
var b = true;
```

You can usually use the global Boolean() function to identify whether an expression is true or false, although it is simpler just to use operators that return Boolean outputs, e.g. `Boolean(2 > 1)`, `(2 > 1)` or even `2 > 1` all return `true`. It is worth noting that the global Boolean() function returns an object and this can in some circumstances behave counterintuitively relative to the primitive Boolean values of `true` and `false`.

Boolean objects support the following properties and methods:

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`length`	Returns the length of a Boolean object	Here
`prototype`	Allows author to add properties and methods to an object	Here

Methods:

Method	Description	More
`toString()`	Converts boolean value to a string	Here
`valueOf()`	Returns the primitive value of the object	Here

Boolean methods:

toString()

[JavaScriptMethodBooleanToString]

The `toString()` method (when applied to a JavaScript Boolean variable) returns a string corresponding to the boolean.

It has the following syntax (with no parameters):

boolean`.toString()`

valueOf()

[JavaScriptMethodBooleanValueOf]

The `valueOf()` method (when applied to a JavaScript Boolean variable) returns the primitive value of the Boolean (i.e. itself).

It has the following syntax (with no parameters):

boolean`.valueOf()`

Appendix U: JavaScript Array Variables

[JavaScriptTutorialArrays]

JavaScript array variables contain multiple (indexed) values in a single variable. Array indices are zero-based, i.e. the first element of the array has as its index 0, the second 1 etc. They are instantiated by statements such as:

```
var a = ["France", "Germany"];
var b = [1, 2, 5, 4];
```

Copying an array is a little more difficult than it looks. This is because each element of an array can itself be an array or an object. For example, the following two assignments don't create two separate arrays.

```
var a = ["France", "Germany"];
var b = a;
```

Instead, if we now set `b[1]` equal to `"Denmark"`, then `a[1]` will also become equal to `"Denmark"`. For the sort of array involved here (e.g. an array of literals then the following will create two separate arrays:

```
var a = ["France", "Germany"];
var b = a.slice();
```

The method `slice()` (with no parameters passed to it) can typically be replaced by `concat()` (also with no parameters passed to it). However, if `a` has some elements that are themselves arrays then the corresponding elements of `b` would still only point to the same physical arrays as elements of a, and changing these would also change the sub-elements in `a`. Also the use of `slice()` and `concat()` may not work as intended if `a` is not an array or is undefined. An alternative that is more robust to unusual forms of arrays involves recursively copying elements, and can be implemented for all types of arrays by adding a method that the array possesses as follows:

```
object.prototype.clone = function() {
  var a = (this instanceof array) ? [] : {};
  for (i in this) {
    if (i == "clone") continue;
    if (this[i] && typeof this[i] == "object") {
      a[i] = this[i].clone();
    }
    else
      a[i] = this[i];
  } return a;
};
```

Arrays support the following properties and methods:

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`length`	Returns length of array	Here
`prototype`	Allows author to add properties and methods to an	Here

	object	

Methods:

Method	Description	More
`concat()`	Joins arrays and returns a copy of the joined array	Here
`copyWithin()`	Copies elements to and from specified positions	Here
`every()`	Returns `true` if every element passes a specified test, otherwise returns `false`	Here
`fill()`	Sets elements of an array to a specified value	Here
`filter()`	Creates a new array consisting only of elements that pass a specified test	Here
`find()`	Returns value of first element that passes a specified test	Here
`findIndex()`	Returns index of first element that passes a specified test	Here
`forEach()`	Calls a function for each element	Here
`indexOf()`	Returns index of first element found when an array is searched	Here
`isArray()`	Returns `true` if object is an array, otherwise `false`	Here
`join()`	Joins all elements of an array into a string	Here
`lastIndexOf()`	Returns index of first element found when an array is searched, backwards from the end	Here
`map()`	Creates a new array consisting of elements which are the result of calling a function on each element in turn	Here
`pop()`	Removes last element of array and returns that element	Here
`push()`	Adds new elements to end of array and returns new length	Here
`reduce()`	Reduces the values of an array to a single value (evaluating from left to right, i.e. from lowest to highest index)	Here
`reduceRight()`	Reduces the values of an array to a single value (evaluating from right to left, i.e. from highest to lowest index)	Here
`reverse()`	Reverses order of array	Here
`shift()`	Removes first element and returns that element	Here
`slice()`	Selects a part of an array and returns that part	Here
`some()`	Returns `true` if at least one element passes a specified test, otherwise returns `false`	Here
`sort()`	Sorts elements of the array	Here
`splice()`	Adds / removes elements	Here
`toString()`	Converts array to a string	Here
`unshift()`	Adds new elements to beginning of the array	Here
`valueOf()`	Returns the primitive value of the object	Here

Array properties:

length

[JavaScriptPropertyArrayLength]

The `length` property (for a JavaScript array) returns the length of the array. An empty array has length 0.

It has the following syntax:

array.`length`

Array methods:

concat()

[JavaScriptMethodArrayConcat]

The `concat()` method (when applied to a JavaScript array) joins arrays and returns a copy of the joined array.

It has the following syntax with the following parameters. It returns an array.

array.`concat(`*array1, array2, …*`)`

Parameter	Required / Optional	Description
array1, array2, …	Required	The arrays to be joined (inserted after the original array in the returned result)

copyWithin()

[JavaScriptMethodArrayCopyWithin]

The `copyWithin()` method (when applied to a JavaScript array) copies elements to and from specified positions.

It has the following syntax with the following parameters. It returns an array (the changed array).

array.`copyWithin(`*target, start, end*`)`

Parameter	Required / Optional	Description
target	Required	The index position where elements start to be copied to
start	Optional	The index position where elements start to be copied from (default is zero)
end	Optional	The index position where elements stop being copied from (default is *array*.`length`

every()

[JavaScriptMethodArrayEvery]

The `every()` method (when applied to a JavaScript array) returns `true` if every element passes a specified test, otherwise returns `false`. It only executes until the function it uses returns a false value and then returns false. It does not execute the function for array elements without values. It does not change the original array

It has the following syntax with the following parameters. It returns a Boolean as above.

array.`every`(*function(currentValue, index, arr), thisValue*)

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element
thisValue	Optional	A value to be passed to the function to be used as its 'this' value (if empty then *thisValue* will be undefined

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

fill()

[JavaScriptMethodArrayFill]

The `fill()` method (when applied to a JavaScript array) sets elements of the array to a specified value.

It has the following syntax with the following parameters. It returns an array (the changed array).

array.`fill`(*value, start, end*)

Parameter	Required / Optional	Description
value	Required	The value to fill the array with
start	Optional	The index position where elements start to be filled (default is zero)
end	Optional	The index position where elements stop being filled (default is *array*.`length`

filter()

[JavaScriptMethodArrayFilter]

The `filter()` method (when applied to a JavaScript array) creates a new array consisting only of elements that pass a specified test. Elements that fail the test are removed.

It has the following syntax with the following parameters. It returns an array as above.

array.`filter`(*function(currentValue, index, arr), thisValue*)

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element
thisValue	Optional	A value to be passed to the function to be used as its 'this' value (if empty then *thisValue* will be undefined

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

find()

[JavaScriptMethodArrayFind]

The `find()` method (when applied to a JavaScript array) returns the value of first element of the array that passes a specified test.

It has the following syntax with the following parameters. It returns an array element as above.

array.`find`(*function(currentValue, index, arr), thisValue*)

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element
thisValue	Optional	A value to be passed to the function to be used as its 'this' value (if empty then *thisValue* will be undefined

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

findIndex()

[JavaScriptMethodArrayFindIndex]

The `findIndex()` method (when applied to a JavaScript array) returns the index of the first element of the array that passes a specified test.

It has the following syntax with the following parameters. It returns a number as above (or -1 if no array element passes the test). It only checks values (i.e. applies the function) up to the first time the test is passed.

array`.findIndex(`*function(currentValue, index, arr), thisValue*`)`

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element
thisValue	Optional	A value to be passed to the function to be used as its 'this' value (if empty then *thisValue* will be undefined

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

forEach()

[JavaScriptMethodArrayForEach]

The `forEach()` method (when applied to a JavaScript array) calls a function for each element of an array that has a value.

It has the following syntax with the following parameters. Its return value is undefined.

array`.forEach(`*function(currentValue, index, arr), thisValue*`)`

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element
thisValue	Optional	A value to be passed to the function to be used as its 'this' value (if empty then *thisValue* will be undefined)

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element

Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

indexOf()

[JavaScriptMethodArrayIndexOf]

The `indexOf()` method (when applied to a JavaScript array) returns the index of the first element of an array found when an array is searched.

It has the following syntax with the following parameters. It returns a number as above (or -1 if no array element passes the test).

array.`indexOf(`*item, start*`)`

Parameter	Required / Optional	Description
item	Required	Item to search for
start	Optional	Index value at which to start search. Negative values indicate start from a position counting back from the end and then search to the end

isArray()

[JavaScriptMethodArrayIsArray]

The `isArray()` method (when applied to the JavaScript Array object) returns `true` if an object is an array, otherwise `false`.

It has the following syntax with the following parameters. It returns a Boolean as above.

`Array.isArray(`*obj*`)`

Parameter	Required / Optional	Description
obj	Required	Item to search for

join()

[JavaScriptMethodArrayJoin]

The `join()` method (when applied to a JavaScript array) joins all elements of an array into a string.

It has the following syntax with the following parameters. It returns a string as above.

array.`join(`*delimiter*`)`

Parameter	Required / Optional	Description

delimiter	Optional	The delimiter (i.e. separator) inserted between consecutive element strings. Default is a comma, i.e. “,”

lastIndexOf()

[JavaScriptMethodArrayLastIndexOf]

The `lastIndexOf()` method (when applied to a JavaScript array) returns the index of the first element of an array found when an array is searched, backwards from the end.

It has the following syntax with the following parameters. It returns a number as above (or -1 if no array element passes the test).

array.`lastIndexOf(`*item, start*`)`

Parameter	Required / Optional	Description
item	Required	Item to search for
start	Optional	Index value at which to start search. Negative values indicate start from a position counting from the beginning and then search backwards to the start

map()

[JavaScriptMethodArrayMap]

The `map()` method (when applied to a JavaScript array) creates a new array consisting of elements which are the result of calling a function on each element of the original array in turn.

It has the following syntax with the following parameters. It returns a new array. It does not execute the function if array element does not have a value and it does not change the original array.

array.`map(`*function(currentValue, index, arr), thisValue*`)`

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element
thisValue	Optional	A value to be passed to the function to be used as its ‘this’ value (if empty then *thisValue* will be undefined

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

pop()

[JavaScriptMethodArrayPop]

The `pop()` method (when applied to a JavaScript array) removes the last element of the array and returns that element.

It has the following syntax with no parameters. It returns the relevant object or primitive that was at the relevant place in the original array.

array.`pop()`

push()

[JavaScriptMethodArrayPush]

The `push()` method (when applied to a JavaScript array) joins arrays and returns a copy of the joined array.

It has the following syntax with the following parameters. It returns a number representing the new length of the array.

array.`push(`*item1, item2, …*`)`

Parameter	Required / Optional	Description
item1, item2, …	Required	The item(s) to be added to the array

reduce()

[JavaScriptMethodArrayReduce]

The `reduce()` method (when applied to a JavaScript array) reduces the values of an array to a single value (from left to right, i.e. from lowest to highest index).

It has the following syntax with the following parameters. It returns the accumulated result from the last call of the function. It does not execute the function if the array element does not have a value.

array.`reduce(`*function(total, currentValue, index, arr), initialValue*`)`

Parameter	Required / Optional	Description
function(total, currentValue, index, arr)	Required	A function to be run for each element
initialValue	Optional	A value to be passed to the function to be used as the initial value

The *function* arguments are:

Parameter	Required / Optional	Description
total	Required	The *initalValue* or the previously returned value of the function
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

reduceRight()

[JavaScriptMethodArrayReduceRight]

The `reduceRight()` method (when applied to a JavaScript array) reduces the values of an array to a single value (evaluating from right to left, i.e. from highest to lowest index).

It has the following syntax with the following parameters. It returns the accumulated result from the last call of the function. It does not execute the function if the array element does not have a value.

array`.reduceRight(`*function(total, currentValue, index, arr), initialValue*`)`

Parameter	Required / Optional	Description
function(total, currentValue, index, arr)	Required	A function to be run for each element
initialValue	Optional	A value to be passed to the function to be used as the initial value

The *function* arguments are:

Parameter	Required / Optional	Description
total	Required	The *initalValue* or the previously returned value of the function
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

reverse()

[JavaScriptMethodArrayReverse]

The `reverse()` method (when applied to a JavaScript array) removes the last element of the array and returns that element.

It has the following syntax with no parameters. It returns an array as above.

array`.reverse()`

shift()

[JavaScriptMethodArrayShift]

The `shift()` method (when applied to a JavaScript array) removes the first element of the array and returns that element.

It has the following syntax with no parameters. It returns the relevant object or primitive that was at the relevant place in the original array.

array.`shift()`

slice()

[JavaScriptMethodArraySlice]

The `slice()` method (when applied to a JavaScript array) selects a part of an array and returns that part.

It has the following syntax with the following parameters. It returns a new array containing the selected elements.

array.`slice(`*start, end*`)`

Parameter	Required / Optional	Description
start	Optional	An integer specifying where to start the selection. Negative numbers are treated as selecting from the end of the array. Default is zero
end	Optional	An integer that specifies where to end the selection. Negative numbers are treated as selecting from the end of the array. Default is to select from *start* to the end of the array

some()

[JavaScriptMethodArraySome]

The `some()` method (when applied to a JavaScript array) returns `true` if at least one element passes a specified test, otherwise returns `false`. It only executes until the function it uses returns a `true` value and then returns `true`. It does not execute the function for array elements without values. It does not change the original array

It has the following syntax with the following parameters. It returns a Boolean as above.

array.`some(`*function(currentValue, index, arr), thisValue*`)`

Parameter	Required / Optional	Description
function(currentValue, index, arr)	Required	A function to be run for each element

thisValue	Optional	A value to be passed to the function to be used as its 'this' value (if empty then *thisValue* will be undefined)

The *function* arguments are:

Parameter	Required / Optional	Description
currentValue	Required	The value of the current element
Index	Optional	The array index of the current element
arr	Optional	The array object which the current element belongs to

sort()

[JavaScriptMethodArraySort]

The `sort()` method (when applied to a JavaScript array) sorts elements of the array.

By default, it sorts values as alphabetical strings in ascending order. This does not work well for numbers

It has the following syntax with the following parameters. It returns a new array containing the selected elements.

array`.sort(`*comparefunction*`)`

Parameter	Required / Optional	Description
comparefunction	Optional	A function that defines an alternative sort order, e.g. `function(a,b){return a - b}` to sort in ascending numerical order or `function(a,b){return b - a}` to sort in descending numerical order

splice()

[JavaScriptMethodArraySplice]

The `splice()` method (when applied to a JavaScript array) adds / removes elements to / from the array.

It has the following syntax with the following parameters. It returns a new array containing the removed items, if any.

array`.splice(`*index, numberremoved, item1, item2, …*`)`

Parameter	Required / Optional	Description
index	Required	Integer specifying position at which to add/remove items. Negative numbers specify

		from end of array
numberremoved	Optional	Number of items to be removed (if set to zero then no items will be removed)
item1, item2, …	Optional	New item(s) to be added to the array

toString()

[JavaScriptMethodArrayToString]

The `toString()` method (when applied to a JavaScript array) returns the string value of the array, with the elements being delimited (separated) by commas.

It has the following syntax (with no parameters):

array`.toString()`

unshift()

[JavaScriptMethodArrayUnshift]

The `unshift()` method (when applied to a JavaScript array) adds new elements to the beginning of the array.

It has the following syntax with the following parameters. It returns a number, representing the new length of the array.

array`.unshift(`*item1, item2, …*`)`

Parameter	**Required / Optional**	**Description**
item1, item2, …	Required	New item(s) to be added to the array

valueOf()

[JavaScriptMethodArrayValueOf]

The `valueOf()` method (when applied to a JavaScript array) returns the primitive value of the array.

It has the following syntax (with no parameters):

array`.valueOf()`

Appendix V: JavaScript Objects

[JavaScriptTutorialObjects]

JavaScript is an object-orientated programming language (like most other more sophisticated general-purpose computer languages) and technically almost all of its components are objects of some sort or other.

JavaScript objects are containers that contain *properties* and *methods*. For example, a statement such as:

```
var person = {title:"Mr", surname:"Smith", age:30}
```

creates an object that has three properties, i.e. name-value, pairs that in this instance characterise (some of the features of) a person.

Object properties can be accessed in two ways, either here e.g. `person.title` or `person["title"]` (both of which in this instance would return a value of `"Mr"`). An array is a specific type of object with the property names indexed from 0 up to the length of the array less 1 (and hence elements of arrays can themselves be arrays (or other sorts of objects).

Object methods are technically also property-like in nature, i.e. again come in name-value pairs, but with the 'name' being a function name and the 'value' being a JavaScript function. For example,

```
<!DOCTYPE html>
<html><head><title>JavaScript Objects</title></head>
<body>
<p>An example of a  JavaScript object</p>
Full Name: <span id="Added"></span><br><br>
Contrast evaluating the function with the contents of the
corresponding property which is:<br>
<span id="Added2"></span>
<script>
window.addEventListener('load', addtext());
function addtext() {
var person = {
   firstName: "John",
   lastName: "Smith",
   fullName: function()
      {return this.firstName + " " + this.lastName}
   }
document.getElementById("Added").innerHTML=person.fullName();
document.getElementById("Added2").innerHTML=person.fullName;
}
</script>
</body>
</html>
```

creates an object with a method called `fullName`. The method is evaluated by calling it as a function (in this case it has no parameters so this involves e.g. `person.fullName()`). In contrast the property fullName (accessed by `person.fullName`, without the ending bracket pair) is the function itself, rather than what the function evaluates to.

Objects have some generic properties, including their constructor, length and prototype properties.

Shared properties applicable to JavaScript objects:

constructor

[JavaScriptPropertyConstructor]

The `constructor` property (when applied to JavaScript object) returns the constructor function for an object (more precisely a reference to that function, rather than the name of the function. Some common functions returned by this property are:

JavaScript variable type	constructor property returns
number	`function Number() { [native code]}`
string	`function String() { [native code]}`
boolean	`function Boolean() { [native code]}`

length

[JavaScriptPropertyLength]

The `length` property (when applied to JavaScript object) returns the length of the object.

It has the following syntax:

object`.length`

Many elements of JavaScript are objects and so can have this property applied to them. For example, applying it to Boolean objects will return 1.

prototype

[JavaScriptPropertyPrototype]

The `prototype` property (when applied to a JavaScript object class) allows the developer to add new properties and methods to that class.

It is a global object constructor available for all JavaScript objects. For example, `Boolean.prototype` does not refer to a single Boolean but to the Boolean() object itself.

It has the following syntax (with no parameters):

ObjectClass`.prototype.`*name* `=` *value*

Where *value* is typically a function definition and *name* is the new method or property to be included in the object class.

Appendix W: JavaScript Error Objects

[JavaScriptTutorialErrorObjects]

JavaScript error objects are used by the try … catch … finally and throw statements to implement structured error handling.

Error objects support the following properties and methods. Note, some browser suppliers e.g. Microsoft have additional non-standard properties such as `.description`, which seems otherwise to be the same as `.message` but will not be recognised by non-Microsoft browsers (so should be avoided if users are likely to use other browsers to view the relevant webpage).

Properties:

Property	Description	More
`constructor`	Returns object's constructor function	Here
`message`	Returns a string containing the error message associated with the error	Here
`name`	Returns the name (i.e. exception type) of an error	Here
`number`	Sets / returns numeric value associated with error	Here
`prototype`	Allows author to add properties and methods to an object	Here
`stack`	Gets trace information regarding the error and is added to when an error is raised, and is updated repeatedly if the error is raised multiple times	Here
`stackTraceLimit`	(Default 10). Sets / returns stack trace limit, i.e. number of error frames displayed in `stack`	Here

Methods:

Method	Description	More
`toString()`	Returns the (string) value of an error object	Here
`valueOf()`	Returns the primitive value of an object	Here

Error properties:

message

[JavaScriptPropertyErrorMessage]

The `message` property (of the JavaScript Error object) returns a string containing the error message associated with the error. The exact result returns varies by browser.

name

[JavaScriptPropertyErrorName]

The `name` property (of the JavaScript Error object) returns the name (i.e. exception type) of an error. When a runtime error occurs then it is set to one of the following native exception types:

Exception type	Description

`ConversionError`	(seems not to be recognised by all browsers). Attempt to convert object into something failed
`RangeError`	Function argument outside its allowable range
`ReferenceError`	Invalid reference detected (e.g. if it is null)
`RegExpError`	(seems not to be recognised by all browsers). Compilation error with a regular expression
`SyntaxError`	When parsed, source text does not follow correct syntax
`TypeError`	Actual type of operand does not match that expected
`URIError`	An illegal URI has been identified, e.g. an illegal character has appeared

number

[JavaScriptPropertyErrorNumber]

The `number` property (of the JavaScript Error object) returns a number characterising the error. This is a 32-bit code, with the upper 16-bits being the facility code and the lower 16-bits being the error code.

stack

[JavaScriptPropertyErrorStack]

The `stack` property (of the JavaScript Error object) returns a string that contains trace information regarding the error. It is added to when an error is raised, and is updated repeatedly if the error is raised multiple times.

stackTraceLimit

[JavaScriptPropertyErrorStackTraceLimit]

The `stackTraceLimit` property (of the JavaScript Error object) sets / returns the stack trace limit, i.e. the number of error frames to display when using the *error*.stack property.

Error methods:

toString()

[JavaScriptMethodErrorToString]

The `toString()` method (when applied to a JavaScript Error object) returns a string representation of the object.

It has the following syntax (with no parameters):

error.`toString()`

valueOf()

[JavaScriptMethodErrorValueOf]

The `valueOf()` method (when applied to a JavaScript Error object) returns the string value of the error.

It has the following syntax (with no parameters):

error.`valueOf()`

Appendix X: JavaScript Operators

[JavaScriptTutorialOperators]

The main operators that JavaScript recognises are set out below:

Arithmetic operators (binary)

Operator	Description	More
+	Addition, e.g. if x is 8 then y = x + 2 results in y becoming 10	Here
–	Subtraction, e.g. if x is 8 then y = x – 2 results in y becoming 6	Here
*	Multiplication, e.g. if x is 8 then y = x * 2 results in y becoming 16	Here
/	Division, e.g. if x is 9 then y = x / 2 results in y becoming 4.5	Here
%	Modulus (i.e. division remainder), e.g. if x is 9 then y = x % 4 results in y becoming 1	Here
**	Exponentiation, if x is 8 then y = x ** 5 results in y becoming 32768	Here

Arithmetic operators (unary)

Operator	Description	More
+	Plus sign, e.g. if x is 8 then then +x represents –8	Here
–	Minus sign, e.g. if x is 8 then then –x represents –8	Here
++	Increment, i.e. add one to the variable. There are technically two different increment operators, the prefix one, and the postfix one. For example, if x is 8 then the statement y = ++x results in x being incremented to 9 and then y assigned this value (i.e. is assigned 9). However, the statement y = x++ involves y being assigned the value of x (i.e. 8) and x then being incremented to 9. For code clarity, some commentators suggest using the statements x = x + 1; y = x; instead of y = ++x and y = x; x = x + 1; instead of y = x++.	Here
--	Decrement, i.e. subtract one from the variable. There are technically two different decrement operators, the prefix one, and the postfix one. For example, if x is 8 then the statement y = --x results in x being decremented to 7 and then y assigned this value (i.e. is assigned 7). However, the statement y = x-- involves y being assigned the value of x (i.e. 8) and x then being decremented to 7. For code clarity, some commentators suggest using the statements x = x - 1; y = x; instead of y = --x and y = x; x = x - 1; instead of y = x--.	Here

Arithmetic Assignment operators

Operator	Description	More
=	Set equal to, e.g. if x is 5 then y = x results in x remaining 5 and y being set to 5	Here
+=	Set equal to after addition, e.g. if x is 5 and y is 8 then y += x results in x remaining 5 and y = y + x, so y becomes 13	Here
-=	Set equal to after subtraction, e.g. if x is 5 and y is 8 then y -= x results in x remaining 5 and y = y - x, so y becomes 3	Here
*=	Set equal to after multiplication, e.g. if x is 5 and y is 8 then y *= x results in x remaining 5 and y = y * x, so y becomes 40	Here
/=	Set equal to after division, e.g. if x is 5 and y is 8 then y /= x results in x remaining 5 and y = y / x, so y becomes 1.6	Here
%=	Set equal to after modulus (i.e. division remainder) e.g. if x is 5 and y is 8 then y %= x results in x remaining 5 and y = y % x, so y becomes 3	Here
**=	Set equal to after exponentiation, e.g. if x is 5 and y is 8 then y **= x results in x remaining 5 and y = y ** x, so y becomes 32768	Here

String operators (binary)

Operator	Description	More
+	Concatenation, e.g. if x is "a" then y = x + "b" results in y becoming "ab"	Here

String Assignment operators

Operator	Description	More
=	Set equal to, e.g. if x is "ab" then y = x results in x remaining "ab" and y being set to "ab"	Here
+=	Set equal to after concatenation, e.g. if x is "a" and y is "b" then y += x results in x remaining "a" and y = y + x, so y becomes "ba"	Here

Comparison operators

Operator	Description	More
==	Equal to, e.g. if x is 8 then x==8 is true, but x==5 is false	Here
===	Equal to and of the same type, e.g. if x is 8 then x===8 is true but x=="8" is false (as not of the same type, although x=="8" would be true, as the computation involves a conversion of "8" to 8)	Here
!=	Not equal to, e.g. if x is 8 then x!=8 is false, but x!=5 is true	Here
!==	Not equal to or not of the same type, e.g. if x is 8 then	Here

	x!== "8" is true (as not of the same type), but x!==8 is false	
>	Greater than, e.g. if x is 8 then x>8 is false, but x>5 is true	Here
<	Less than, e.g. if x is 8 then x<8 is false, but x<11 is true	Here
>=	Greater than or equal to, e.g. if x is 8 then x>=8 is true, but x>11 is false	Here
<=	Less than or equal to, e.g. if x is 8 then x<=8 is true, but x<5 is false	Here

Conditional (ternary, i.e. IIF) operator

Operator	Description	More
var3 = (*boolval*) ? *var1* : *var2*	Sets *var3* equal to *var1* if *boolval* is true, otherwise sets *var3* equal to *var2*	Here

Boolean (i.e. logical) operators

Operator	Description	More
`&&`	Boolean AND operator, e.g. if `x` is `5` and `y` is `8` then `(x<6 && y>7)` is `(true && true)`, i.e. is `true`	Here
`||`	Boolean OR operator, e.g. if `x` is `5` and `y` is `8` then `(x<6 && y>10)` is `(true || false)`, i.e. is `true`	Here
`!`	Boolean NOT operator, e.g. if `x` is 5 then `!(x==5)` is `!true`, i.e. is `false`	Here

Bitwise operators

In JavaScript, these operators apply to (signed) 32-bit numbers. They are particularly fast to compute, because they have very simple analogues in how most computer central processing units (CPUs) operate. However, they can be less easy to follow than using corresponding primitive mathematical operators such as division and modulus remainder (as the latter are not base-2 specific). Application of the bitwise NOT and XOR operators can also be somewhat counter-intuitive for small unsigned numbers because of their application to signed 32-bit numbers.

Operator	Description	More
`&`	Bitwise AND operator, e.g. `6` & `2` in binary notation is `110 & 010` so is `010`, i.e. `2` in decimal notation	Here
`|`	Bitwise OR operator, e.g. `6` | `2` in binary notation is `110 | 010` so is `110`, i.e. `6` in decimal notation	Here
`~`	Bitwise NOT operator, e.g. ~ `6` in binary notation is ~ `110` so is `1(29 times)001`, which is interpreted as a negative number so is treated as −`7` in decimal notation	Here
`^`	Bitwise XOR operator	Here
`<<`	Bitwise left shift operator, e.g. `6 << 2` in binary notation is `110` shifted `2` bits to left so is `11000`, i.e. `24` in decimal notation	Here
`>>`	Bitwise right shift operator, e.g. `6 >> 2` in binary notation is `110` shifted `2` bits to the right so is `1`, i.e. `1` in decimal notation	Here

delete, in, instanceof, typeof, yield, yield* and void operators

Operator	Description	More
`delete`	Deletes a property from an object	Here
`in`	Returns true if the specified property is in the specified object, otherwise false.	Here
`instanceof`	Returns `true` if a specified object is an instance of the specified object type, otherwise returns `false`	Here
`typeof`	Returns the type of a variable, object, function or expression	Here
`yield`		
`yield*`		
`void`	Evaluates an expression and then returns undefined	Here

Spread and comma/sequence operators

Operator	Description	More
...	Allows an iterable such as an array expression to be expanded where zero or more arguments or elements are expected	
,	Evaluates each operand from left to right and returns the value of the last operand (e.g. in a `for` loop). It is not the same as the comma within arrays, objects and function arguments and parameters	

Operator precedence

When an expression includes more than one operator it becomes important to identify the precedence given to different operators, i.e. which one is executed first. We set out here details of operator precedence used in JavaScript, as well as some of the more unusual expression components that are formally considered to be operators in the JavaScript (i.e. ECMAScript) specification.

JavaScript Operator Precedence

[JavaScriptOperatorPrecedence]

JavaScript adopts the following operator precedence (which is broadly in line with most other main-stream object orientated programming languages). In this list earlier means higher precedence (i.e. applied before later operators on the list), with operators with equal precedence grouped together:

Precedence Level	Operator	Associativity	Description	Example
20	`( )`	N/A	Expression grouping (brackets)	`(1+2)`
19	`.`	Left-to-right	Member of (object)	`country.state`
	`[]`	Left-to-right	Member of (array)	`country["state"]`
	`()`	Left-to-right	Function call	`testFunction()`
	`new` (with argument list)	N/A	Create statement / operator	`new Date()`
18	`new` (without	Right-to-left	Create statement /	`new`

	argument list)		operator	
17	++	N/A	Postfix increment	x++
	--	N/A	Postfix decrement	x--
16	++	Right-to-left	Prefix increment	++y
	--	Right-to-left	Prefix decrement	--y
	+ (unary)	Right-to-left	Unary plus	+y
	- (unary)	Right-to-left	Unary minus	-y
	!	Right-to-left	Logical not	!(x==20)
	~	Right-to-left	Bitwise not	~5
	typeof	Right-to-left	typeof operator	typeof x
	void	Right-to-left	void operator	void()
	delete	Right-to-left	Delete operator	delete x.value
15	**	Right-to-left	Exponentiation (older browsers do not necessarily support this operator)	10 ** 2
14	*	Left-to-right	Multiplication	4 * 3
	/	Left-to-right	Division	6 / 3
	%	Left-to-right	Modulo division	7 / 5
13	+	Left-to-right	Addition (or concatenation)	4 + 3
	-	Left-to-right	Subtraction	4 - 8
12	<<	Left-to-right	Bitwise shift left	10 << 2
	>>	Left-to-right	Bitwise shift right	10 >> 2
	>>>	Left-to-right	Bitwise unsigned shift right	-10 >> 2
11	<	Left-to-right	Less than	x < y
	<=	Left-to-right	Less than or equal	x <= y
	>	Left-to-right	Greater than	x > y
	>=	Left-to-right	Greater than or equal	x >= y
	in	Left-to-right	in operator	x in y
	instanceof	Left-to-right	instanceof operator	x instance of Array
10	==	Left-to-right	Equal	x == y
	===	Left-to-right	Strictly equal	x === y
	!=	Left-to-right	Unequal	x != y
	!==	Left-to-right	Strictly unequal	x !== y
9	&	Left-to-right	Bitwise AND	x & y
8	^	Left-to-right	Bitwise XOR	x ^ y
7	|	Left-to-right	Bitwise OR	x | y
6	&&	Left-to-right	Logical AND	x && y
5	||	Left-to-right	Logical OR	x || y
4	? :	Right-to-left	Conditional	bool ? y : z
3	=	Right-to-left	Assignment	x = y
	+=	Right-to-left	Assignment with addition (or concatenation)	x += y
	-=	Right-to-left	Assignment with subtraction	x *= y
	**=	Right-to-left	Assignment with	x **= y

			exponentiation	
	`*=`	Right-to-left	Assignment with multiplication	`x *= y`
	`/=`	Right-to-left	Assignment with division	`x /= y`
	`%=`	Right-to-left	Assignment with modulus	`x %= y`
	`<<=`	Right-to-left	Assignment with left shift	`x <<= y`
	`>>=`	Right-to-left	Assignment with right shift	`x >>= y`
	`>>>=`	Right-to-left	Assignment with unsigned right shift	`x >>>= y`
	`&=`	Right-to-left	Assignment with bitwise AND	`x &= y`
	`^=`	Right-to-left	Assignment with bitwise XOR	`x ^= y`
	`|=`	Right-to-left	Assignment with bitwise XOR	`x |= y`
2	`yield`	Right-to-left		`yield x`
	`yield*`	Right-to-left		`yield*`
1	…	N/A		… `x`
0	`,`	Left-to-right		`x, y, z`

The associativity determines the order in which operators of the same precedence are processed. Left-to-right means that a OP b OP c is processed as (a OP b) OP c. Right-to-left means that a OP b OP c is processed as a OP (b OP c).

Assignment operators are right-associative (i.e. right-to-left) so that `a = b = 4` will result in both `a` and `b` being given the value of `4`. This is because the assignment operator returns the value that is assigned, i.e. `(b=4)` returns `4`, which is then assigned to `a`.

JavaScript Coercion and 'Typing'

[JavaScriptTutorialCoercion]

Introduction

Coercion and associated '**typing**' concepts are some of the subtler aspects of JavaScript. They can result in code that appears to exhibit counterintuitive behaviour. In this page, we summarise some of the complexities involved and their implications for software development.

Variable types

Most programming languages include the concept of variables and types that such variables can exhibit. For example, string variables (text) are usually differentiated from numerical variables (numbers), since the sorts of manipulations that can typically be applied to strings differ from the ones that can typically be applied to numbers.

JavaScript has 5 primitive data types as set out below, as well as some other types, such as objects:

Data type	Purpose
number	Numbers
string	Strings
boolean	True / False
null	Empty values
undefined	Variable declared but not yet assigned a value

Weakly typed versus strongly typed software languages

Some (object orientated) programming languages require the software developer to specify what type a variable can take and then the variable is required to stay of that type. These are called '**strongly typed**' languages. JavaScript adopts a different approach; its variables are '**weakly typed**'. Their type can change part way through software execution. For example, the following statements could be included in JavaScript without throwing an error, and when executed would leave `x` equal to "hello".

```
var x = 14; x="hello";
```

Even languages that are mostly strongly typed may possess 'variant' types (e.g. VisualBasic) which can in effect change their underlying type as execution progresses. In such languages, variables may be defaulted to have these types if no other type is specified when they are declared. Such programming languages often have means of forcing developers to specify the type that a variable takes (even if just that it is a 'variant'), e.g. the `option explicit` command in VisualBasic, since doing so can lead to more robust code.

Some commentators view strong typing and weak typing as synonymous with compiled versus interpreted computer language. Classically, a compiled language is one in which the software is converted into a machine-code executable before running, whilst an interpreted language is one in which the software statements are analysed and executed 'on-the-fly' as the program executes. Machine code generally only functions on primitive data types so ultimately requires some form of strong typing. However, nowadays the previous gulf between compiled and interpreted languages has declined, with e.g. 'just-in-time' compilation and other clever compiling / interpreting techniques. The concept of a 'variant' type, an object that includes as one of its properties the type of data that it currently represents, also diminishes the difference between these two language types.

Type coercion

If you apply operators to variables (or values) that do not immediately make sense with such operators, e.g. trying to add a string to a number, then JavaScript will typically try to make sense of the expression by carrying out **type coercions**. These involve converting the type of one or more expression components into new types that do make sense in the context of the operator.

Potentially counterintuitive behaviours

The weak typing used in JavaScript can lead to potentially counterintuitive behaviours or errors, since the type coercion carried out by JavaScript may not always be as the developer expects. Conversely, it also sometimes makes for simpler code. This is because some expressions that might otherwise not be expected to return meaningful answers do so in ways that can be helpful to program logic.

Examples of potentially counterintuitive behaviours include ones where text can be interpreted as numbers and vice versa. For example, if either of the parameters to which a + operator is applied is a string then the other will where necessary be coerced to a string. The exact result of an expression can then depend on the order in which the operations are applied. This depends on the precedence given to different operators, see here. For example:

```
var x = 1 + 2 + "5";
var y = (1 + 2) + "5";
var z = 1 + (2 + "5");
```

result in `x` = `"35"`, `y` = `"35"` and `z` = `"125"`

Individual JavaScript Operators

assignment

[JavaScriptOperatorAssignment]

In JavaScript, the = operator is the assignment operator.

For example, if `x` is `5` then `y = x` results in `x` remaining `5` and `y` being set to `5`, or if `x` is `"b"` then `y = x` results in `y` becoming `"b"` too.

bitwise AND

[JavaScriptOperatorBitwiseAnd]

In JavaScript, the `&` operator is the bitwise AND operator.

For example, `6` & `2` in binary notation is `110 & 010` so is `010`, i.e. `2` in decimal notation.

bitwise left shift

[JavaScriptOperatorBitwiseLeftShift]

In JavaScript, the << operator is the bitwise left shift operator.

For example, `6 << 2` in binary notation is `110` shifted `2` bits to the left so is `11000`, i.e. `24` in decimal notation.

bitwise NOT

[JavaScriptOperatorBitwiseNot]

In JavaScript, the ~ operator is the bitwise OR operator.

In JavaScript, the ~ operator is the bitwise OR operator.

For example, ~ `6` in binary notation is ~ `110` so is `1(29 times)001`, which is interpreted as a negative number so is treated as −`7` in decimal notation.

bitwise OR

[JavaScriptOperatorBitwiseOr]

In JavaScript, the `|` operator is the bitwise OR operator.

For example, `6 | 2` in binary notation is `110 | 010` so is `110`, i.e. `6` in decimal notation.

bitwise right shift

[JavaScriptOperatorBitwiseRightShift]

In JavaScript, the `>>` operator is the bitwise right shift operator.

For example, `6 >> 2` in binary notation is `110` shifted `2` bits to the right so is `1`, i.e. `1` in decimal notation.

bitwise XOR

[JavaScriptOperatorBitwiseXor]

In JavaScript, the `^` operator is the bitwise XOR operator.

For example, `6 ^ 2` in binary notation is `110 ^ 010` so is `100`, i.e. `4` in decimal notation.

conditional

[JavaScriptOperatorConditional]

In JavaScript, the conditional operator is the (ternary) operator that selects between two possible expressions to evaluate depending on whether an expression is `true` or `false`.

For example, `var3 = (boolval) ? var1 : var2` results in `var3` being set equal to `var1` if `boolval` is `true`, otherwise `var3` is set equal to `var2`.

decrement

[JavaScriptOperatorDecrement]

In JavaScript, the `--` operator is the (unary) arithmetic operator for decrementing, i.e. subtracting one from the variable.

There are technically two different decrement operators, the prefix one, and the postfix one. For example, if `x` is `8` then the statement `y = --x` results in `x` being decremented to `7` and then y assigned this value (i.e. is assigned `7`). However, the statement `y = x--` involves `y` being assigned the value of `x` (i.e. `8`) and `x` then being decremented to `7`.

For code clarity, some commentators suggest using the statements `x = x - 1; y = x;` instead of `y = --x` and `y = x; x = x - 1;` instead of `y = x--`.

delete

[JavaScriptOperatorDelete]

In JavaScript, the `delete` operator deletes a property from an object. For example, suppose:

```
var x = {name: "ten", val: 10, binaryval: "1010"}
```

This creates an object with three name/value pairs.

Then `delete x.value` (or `delete x["val"]`) would result in `x` becoming:

```
{name: "ten", binaryval: "1010"}
```

It is usually unwise to apply the delete operator to predefined JavaScript object properties, as otherwise the application can crash.

divide

[JavaScriptOperatorDivide]

In JavaScript, the `/` operator is the (binary) arithmetic operator for division.

For example, if `x` is `9` then `y = x / 2` results in `y` becoming `4.5`.

divide assignment

[JavaScriptOperatorDivideAssignment]

In JavaScript, the `/=` operator is the assignment operator with division.

For example, if `x` is `5` and `y` is `8` then `y /= x` results in `x` remaining `5` and `y = y / x`, so `y` becomes `1.6`.

equal

[JavaScriptOperatorEqual]

In JavaScript, the `==` operator is the 'equal to' operator:

x	y	x == y
8	8	true
8	5	false

Note: if `x` and `y` are of different type then some type coercion will typically occur, and values that developers might not immediately recognise as 'equal' may be deemed equal by this operator. If you want only variables that are of the same type to pass the equality test then you should use the strictly equal to operator, i.e. ===.

exponentiation

[JavaScriptOperatorExponentiation]

In JavaScript, the `**` operator is the (binary) arithmetic operator for exponentiation.

For example, if `x` is `8` then `y = x ** 5` results in `y` becoming `32768`.

Note: older browsers may not recognise this operator or the **= operator, in which case the Math.pow() method would need to be used.

exponentiation assignment

[JavaScriptOperatorExponentiationAssignment]

In JavaScript, the `**=` operator is the assignment operator with exponentiation.

For example, if `x` is `5` and `y` is `8` then `y **= x` results in `x` remaining `5` and `y = y ** x`, so `y` becomes `32768`.

Note: older browsers may not recognise this operator or the ** operator, in which case the Math.pow() method would need to be used.

greater than

[JavaScriptOperatorGreaterThan]

In JavaScript, the > operator is the 'greater than' operator, e.g.:

x	y	x > y
8	8	false
8	5	true

greater than or equal

[JavaScriptOperatorGreaterThanOrEqual]

In JavaScript, the >= operator is the 'greater than or equal' operator, e.g.:

x	y	x >= y
8	8	true
8	11	false

in

[JavaScriptOperatorIn]

In JavaScript, the `in` operator returns true if the specified property is in the specified object, otherwise false. The results can be a little counterintuitive until the exact nature of the object is taken into account:

For example, suppose we have an object such as:

```
var x = {name: "ten", val: 10, binaryval: "1010"};
```

Then:

y	y in x	Explanation
`"name"`	true	"name" is a valid property of the object
`"val"`	true	"val" is a valid property of the object
`"length"`	true	The object has a length (number of entries)
`1`	false	An object is not indexed like an array, so there is no entry indexed by 1

Suppose, instead, we have an array such as:

```
var x = ["a","b"];
```

Then

y	y in x	Explanation
"a"	false	"a" is not a valid property, rather it is the value assigned to the property (the property is the index number)
"length"	true	As arrays do generically have a length property
1	true	As there is an entry with index number 1 (the entry is "b"), although if y were 2 then it would be false (as there is no entry with index number 2)

increment

[JavaScriptOperatorIncrement]

In JavaScript, the `++` operator is the (unary) arithmetic operator for incrementing, i.e. adding one to the variable. There are technically two different increment operators, the prefix one, and the postfix one.

For example, if `x` is `8` then the statement `y = ++x` results in `x` being incremented to `9` and then y assigned this value (i.e. is assigned `9`). However, the statement `y = x++` involves `y` being assigned the value of `x` (i.e. `8`) and `x` then being incremented to `9`.

For code clarity, some commentators suggest using the statements `x = x + 1; y = x;` instead of `y = ++x` and `y = x; x = x + 1;` instead of `y = x++`.

instanceof

[JavaScriptOperatorInstanceof]

In JavaScript, the `instanceof` operator returns `true` if a specified object is an instance of the specified object type, otherwise returns `false`.

For example, suppose we have an array:

```
var x = ["a","b"];
```

Then `(x instanceof Array)` and `(x instanceof Object)` both return `true` (because `x` is an array and arrays are themselves objects, but `(x instanceof String)` returns false

less than

[JavaScriptOperatorLessThan]

In JavaScript, the < operator is the 'less than' operator, e.g:

x	y	x < y
8	8	false
8	11	true

less than or equal

[JavaScriptOperatorLessThanOrEqual]

In JavaScript, the <= operator is the 'less than or equal' operator, e.g:

x	y	x <= y
8	8	true
8	5	false

logical AND

[JavaScriptOperatorLogicalAnd]

In JavaScript, the `&&` operator is the logical (i.e. Boolean) AND operator:

x	y	x AND y
true	true	true
true	false	false
false	true	false
false	false	false

logical NOT

[JavaScriptOperatorLogicalNot]

In JavaScript, the `!` operator is the logical (i.e. Boolean) NOT operator:

x	!x
true	false
false	true

logical OR

[JavaScriptOperatorLogicalOr]

In JavaScript, the `||` operator is the logical (i.e. Boolean) OR operator:

x	y	x OR y
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`

minus

[JavaScriptOperatorMinus]

In JavaScript, the `-` operator has two possible meanings:

Arithmetic operator (binary)

Subtraction, e.g. if `x` is `8` then `y = x - 2` results in `y` becoming `6`

Arithmetic operator (unary)

Minus sign, e.g. if `x` is `8` then then `-x` represents `-8`

minus assignment

[JavaScriptOperatorMinusAssignment]

In JavaScript, the `-=` operator is the assignment operator with subtraction.

For example, if `x` is `5` and `y` is `8` then `y -= x` results in `x` remaining `5` and `y = y - x`, so `y` becomes `3`.

modulus

[JavaScriptOperatorModulus]

In JavaScript, the `%` operator is the (binary) arithmetic operator for modulus (i.e. division remainder).

For example, if `x` is `9` then `y = x % 4` results in `y` becoming `1`.

modulus assignment

[JavaScriptOperatorModulusAssignment]

In JavaScript, the `%=` operator is the assignment operator with modulus (i.e. division remainder).

For example, if `x` is `5` and `y` is `8` then `y %= x` results in `x` remaining `5` and `y = y % x`, so `y` becomes `3`.

multiply

[JavaScriptOperatorMultiply]

In JavaScript, the `*` operator is the (binary) arithmetic operator for multiplication.

For example, if `x` is `8` then `y = x * 2` results in `y` becoming `16`.

multiply assignment

[JavaScriptOperatorMultiplyAssignment]

In JavaScript, the `*=` operator is the assignment operator with multiplication.

For example, if `x` is `5` and `y` is `8` then `y *= x` results in `x` remaining `5` and `y = y * x`, so `y` becomes `40`.

not equal

[JavaScriptOperatorNotEqual]

In JavaScript, the `!=` operator is the 'not equal to' operator:

x	y	x != y
8	8	`false`
8	5	`true`

Note: if `x` and `y` are of different type then some type coercion will typically occur, and values that developers might not immediately recognise as 'unequal' may be deemed unequal by this operator. If you want only variables that are of the same type to pass the equality test then you should use the strictly unequal to operator, i.e. `!==`.

plus

[JavaScriptOperatorPlus]

In JavaScript, the `+` operator has three possible meanings:

Arithmetic operator (binary)

Addition, e.g. if `x` is `8` then `y = x + 2` results in `y` becoming `10`

Arithmetic operator (unary)

Plus sign, e.g. if `x` is `8` then then `+x` represents `-8`

String operator (binary)

Concatenation, e.g. if `x` is `"a"` then `y = x + "b"` results in `y` becoming `"ab"`

Further comments

If an expression involves 'adding' a string to a number then the number is coerced to a string and string concatenation is applied.

plus assignment

[JavaScriptOperatorPlusAssignment]

In JavaScript, the `+=` operator is the assignment operator with addition (if arithmetic) or concatenation (if string).

Arithmetic operator

For example, if `x` is `5` and `y` is `8` then `y += x` results in `x` remaining `5` and `y = y + x`, so `y` becomes `13`.

String operator

For example, if `x` is `"a"` and `y` is `"b"` then `y += x` results in `x` remaining `"a"` and `y = y + x`, so `y` becomes `"ba"`.

strictly equal

[JavaScriptOperatorStrictlyEqual]

In JavaScript, the `===` operator is the 'strictly equal to' operator:

x	y	x == y
8	"8"	false
8	8	true

Note: if `x` and `y` are of the same type then this operator should return the same as the 'equal to' operator, `==`.

strictly not equal

[JavaScriptOperatorStrictlyNotEqual]

In JavaScript, the `!==` operator is the 'strictly not equal to' operator:

x	Y	x != y
8	8	false
8	5	true

Note: if `x` and `y` are of the same type then this operator should return the same as the 'not equal to' operator, `!=`.

typeof

[JavaScriptOperatorTypeof]

In JavaScript, the `typeof` operator returns the type of a variable, object, function or expression, e.g.

x	typeof x
`false`	`boolean`
`true`	`boolean`
`0`	`number`
`1`	`number`
`"0"`	`string`
`"1"`	`string`
`NaN`	`number`
`Infinity`	`[???]`
`-Infinity`	`[???]`
`""`	`string`
`"10"`	`string`
`"ten"`	`string`
`[ ]`	`object`
`[10]`	`object`
`[5,10]`	`object`
`["ten"]`	`object`
`["five","ten"]`	`object`
`function(){}`	`function`
`{ }`	`object`
`null`	`object`
`undefined`	`undefined`

The type of a date, array or null is an object, which means that you cannot directly use the `typeof` operator to work out if a JavaScript object is an array rather than a date, say.

void

[JavaScriptOperatorVoid]

In JavaScript, the `void` operator evaluates an expression and then returns `undefined`. An example of its use might be e.g.:

```
<a href="javascript: void(document.getElementById('xxx').
innerHTML='Changed text'); ">
Click to change text in element with id = xxx
</a>
```

Appendix Y: The JavaScript Document Object Model (DOM)

[JavaScriptDOM]

The JavaScript DOM is summarised here. Details of individual parts of the DOM (and the associated BOM) are set out below:

1. DOM own properties and methods

[JavaScriptTutorialDOMDetails1]

The JavaScript DOM (`document` object) supports the following (own) properties and methods:

Properties:

Property	Description	More
`activeElement`	Returns the element that currently has focus	Here
`anchors`	Returns collection of all <a> elements that have a name attribute	Here
`applets`	Returns collection of all <applet> elements that have a name attribute	Here
`baseURI`	Returns absolute base URI	Here
`body`	Returns the <body> element	Here
`cookie`	Returns all name/value cookie pairs	Here
`characterSet`	Returns character encoding	Here
`charset`	Depreciated (use `characterSet` instead). Returns character encoding	Here
`doctype`	Returns document type	Here
`documentElement`	Returns main document element of the document (i.e. its <html> element)	Here

documentMode	Returns mode used by browser to render document	Here
domain	Returns domain name of server	Here
domConfig	Obsolete. Returns DOM configuration	Here
embeds	Returns collection of all <embed> elements	Here
forms	Returns collection of all <form> elements	Here
head	Returns the <head> element	Here
images	Returns collection of all <img> elements	Here
implementation	Returns DOMImplementation object handling document	Here
inputEncoding	Returns encoding (character set) used for document	Here
lastModified	Returns date and time document last modified	Here
links	Returns collection of all <a> and <area> elements that have a href attribute	Here
readyState	Returns load status of the document	Here
referrer	Returns URL of the document that loaded the current document	Here
scripts	Returns collection of all <script> elements	Here
strictErrorChecking	Sets / returns whether to enforce strict error checking	Here
title	Sets / returns document <title>	Here
URL	Returns full URL	Here

Methods:

Method	Description	More
addEventListener()	Attaches an event handler	Here
adoptNode()	Adopts a node from another document	Here
close()	Closes output stream previously opened using open()	Here
createAttribute()	Creates an attribute node	Here
createComment()	Creates a comment node	Here
createDocumentFragment()	Creates an empty DocumentFragment node	Here
createElement()	Creates an element node	Here
createTextNode()	Creates a text node	Here
getElementById()	Returns element with specified id attribute	Here
getElementsByClassName()	Returns NodeList containing all elements with specified class attribute	Here
getElementsByName()	Returns NodeList containing all elements with specified name attribute	Here
getElementsByTagName()	Returns NodeList containing all elements with specified tag name (i.e. element type)	Here
hasFocus()	Returns true if document has focus, otherwise returns false	Here
importNode()	Imports node from another document	Here
normalize()	Removes empty text nodes and joins adjacent notes	Here
normalizeDocument()	Removes empty text nodes and joins adjacent notes	Here
open()	Opens an HTML output stream (into which output from write() or writeln() can	Here

	go)	
`querySelector()`	Returns first (child) element that matches specified CSSSelector	Here
`querySelectorAll()`	Returns a NodeList (collection) containing all (child) elements that match specified CSSSelector(s)	Here
`removeEventListener()`	Detaches (removes) an event handler	Here
`renameNode()`	Renames specified node	Here
`write()`	Writes HTML (which can include JavaScript code) to the document	Here
`writeln()`	As per `write()` except that it adds a new line character after each statement	Here

Further comments:

The `document` object also supports some generic properties and methods that can be used on all HTML elements / nodes, even though several of them have no natural meaning when applied to the `document` object.

JavaScript DOM own properties:

activeElement

[JavaScriptPropertyDomActiveElement]

The `activeElement` property of the JavaScript DOM returns the HTML element that currently has focus.

anchors

[JavaScriptPropertyDomAnchors]

The `anchors` property of the JavaScript DOM returns a collection of all <a> elements that have a name attribute.

applets

[JavaScriptPropertyDomApplets]

The `applets` property of the JavaScript DOM returns a collection of all <applet> elements that have a name attribute.

baseURI

[JavaScriptPropertyDomBaseURI]

The `baseURI` property of the JavaScript DOM returns the absolute base URI of the document.

body

[JavaScriptPropertyDomBody]

The `body` property of the JavaScript DOM returns the <body> element of the document.

characterSet

[JavaScriptPropertyDomCharacterSet]

The `characterSet` property of the JavaScript DOM returns the character encoding of the document.

charset

[JavaScriptPropertyDomCharset]

The `charset` property of the JavaScript DOM is depreciated. Use the characterSet property instead. It returns the character encoding of the document.

cookie

[JavaScriptPropertyDomCookie]

The `cookie` property of the JavaScript DOM returns all name/value cookie pairs.

doctype

[JavaScriptPropertyDomDoctype]

The `doctype` property of the JavaScript DOM returns a DocumentType object specifying the doctype of the document. The DocumentType object in turn has a `name` property which returns the name of the doctype.

documentElement

[JavaScriptPropertyDomDocumentElement]

The `documentElement` property of the JavaScript DOM returns the main document element of the document (i.e. its <html> element).

documentMode

[JavaScriptPropertyDomDocumentMode]

The `documentMode` property of the JavaScript DOM returns the mode used by the browser to render the document.

domain

[JavaScriptPropertyDomDomain]

The `domain` property of the JavaScript DOM returns the domain name of the server.

domConfig

[JavaScriptPropertyDomDomConfig]

The `domConfig` property of the JavaScript DOM is obsolete. It returns the DOM configuration.

embeds

[JavaScriptPropertyDomEmbeds]

The `embeds` property of the JavaScript DOM returns a collection of all the <embed> elements in the document.

forms

[JavaScriptPropertyDomForms]

The `forms` property of the JavaScript DOM returns a collection of all the <form> elements in the document.

head

[JavaScriptPropertyDomHead]

The `head` property of the JavaScript DOM returns the <head> element of the document.

images

[JavaScriptPropertyDomImages]

The `images` property of the JavaScript DOM returns a collection of all the <img> elements in the document.

implementation

[JavaScriptPropertyDomImplementation]

The `implementation` property of the JavaScript DOM returns the DOMImplementation object handling document.

inputEncoding

[JavaScriptPropertyDomInputEncoding]

The `inputEncoding` property of the JavaScript DOM returns the encoding (i.e. character set) used for the document.

lastModified

[JavaScriptPropertyDomLastModified]

The `lastModified` property of the JavaScript DOM returns the date and time the document was last modified.

links

[JavaScriptPropertyDomLinks]

The `links` property of the JavaScript DOM returns a collection of all the <a> and <area> elements in the document that have a href attribute.

readyState

[JavaScriptPropertyDomReadyState]

The `readyState` property of the JavaScript DOM returns the load status of the document.

referrer

[JavaScriptPropertyDomReferrer]

The `referrer` property of the JavaScript DOM returns the URL of the document that loaded the current document.

scripts

[JavaScriptPropertyDomScripts]

The `scripts` property of the JavaScript DOM returns a collection of all the <script> elements in the document.

strictErrorChecking

[JavaScriptPropertyDomStrictErrorChecking]

The `strictErrorChecking` property of the JavaScript DOM sets / returns whether to enforce strict error checking of the document.

title

[JavaScriptPropertyDomTitle]

The `title` property of the JavaScript DOM sets / returns the document <title>.

URL

[JavaScriptPropertyDomURL]

The `URL` property of the JavaScript DOM returns the full URL of the document.

JavaScript DOM own methods:

addEventListener()

[JavaScriptMethodDomAddEventListener]

The `addEventListener()` method (when applied to the document object of the JavaScript DOM) attaches an event handler to the document.

It has the following syntax with the following parameters. It does not return any value.

`document.addEventListener(`*event*`,` *function*`,` *useCapture*`)`

Parameter	Required / Optional	Description
event	Required	String specifying event (excluding the '`on`' part at the start of the relevant event attribute name)
function	Required	Name of function (with '`()`' included at end)
useCapture	Optional	If `true` then event handler is executed in the capturing phase of the page load, if false then in the bubbling phase

Some earlier versions of some major browsers do not support this method. For these browsers you instead need to use the `attachEvent()` method.

adoptNode()

[JavaScriptMethodDomAdoptNode]

The `adoptNode()` method (when applied to the document object of the JavaScript DOM) adopts a node from another document.

It has the following syntax with the following parameters. It returns a node object, representing the adopted node.

`document.adoptNode(`*node*`)`

Parameter	Required / Optional	Description
node	Required	Node object (of any type) from another node

close()

[JavaScriptMethodDomClose]

The `close()` method (when applied to the document object of the JavaScript DOM) closes the output stream previously opened using `document.open()`.

It has the following syntax. It takes no parameters and does not return a value.

```
document.close(node)
```

createAttribute()

[JavaScriptMethodDomCreateAttribute]

The `createAttribute()` method (when applied to the document object of the JavaScript DOM) creates an attribute node.

It has the following syntax with the following parameters. It returns a node object represents the created attribute.

```
document.createAttribute(attributename)
```

Parameter	Required / Optional	Description
attributename	Required	An Attr object which is the name of the attribute to create

createComment()

[JavaScriptMethodDomCreateComment]

The `createComment()` method (when applied to the document object of the JavaScript DOM) creates a comment node.

It has the following syntax with the following parameters. It returns a comment object.

```
document.createComment(text)
```

Parameter	Required / Optional	Description
text	Optional	The text to include in the comment object

createDocumentFragment()

[JavaScriptMethodDomCreateDocumentFragment]

The `createDocumentFragment()` method (when applied to the document object of the JavaScript DOM) creates an empty DocumentFragment node.

It has the following syntax with no parameters. It returns an empty DocumentFragment node.

```
document.createDocumentFragment()
```

createElement()

[JavaScriptMethodDomCreateElement]

The `createElement()` method (when applied to the document object of the JavaScript DOM) creates a HTML element node.

It has the following syntax with the following parameters. It returns an element object, representing the created element.

`document.createElement(`*nodetype*`)`

Parameter	Required / Optional	Description
nodetype	Required	String specifying the name (type) of element to be created

createTextNode()

[JavaScriptMethodDomCreateTextNode]

The `createTextNode()` method (when applied to the document object of the JavaScript DOM) creates a text node.

It has the following syntax with the following parameters. It returns an element object, representing the created element.

`document.createTextNode(`*text*`)`

Parameter	Required / Optional	Description
text	Required	String specifying the text in the text node

getElementById()

[JavaScriptMethodDomGetElementById]

The `getElementById()` method (when applied to the document object of the JavaScript DOM) returns the element with the specified id attribute (if it exists).

It has the following syntax with the following parameters. It returns an element object, representing the element with the specified *id* as its id attribute or null if no such element exists.

`document.getElementById(`*id*`)`

Parameter	Required / Optional	Description
id	Required	String specifying the value of the id attribute of the element you want to obtain

getElementsByClassName()

[JavaScriptMethodDomGetElementsByClassName]

The `getElementsByClassName()` method (when applied to the document object of the JavaScript DOM) returns a NodeList containing all the elements with the specified class attribute.

It has the following syntax with the following parameters. It returns a NodeList representing a collection of all relevant elements, ordered as they appear in the source code.

```
document.getElementsByClassName(classname)
```

Parameter	Required / Optional	Description
classname	Required	String specifying the class name of the elements you want to obtain. To include multiple class names, separate individual class names by spaces

getElementsByName()

[JavaScriptMethodDomGetElementsByName]

The `getElementsByName()` method (when applied to the document object of the JavaScript DOM) returns a NodeList containing all the elements with the specified name attribute.

It has the following syntax with the following parameters. It returns a NodeList representing a collection of all relevant elements, ordered as they appear in the source code.

```
document.getElementsByName(name)
```

Parameter	Required / Optional	Description
name	Required	String specifying the name attribute value of the elements you want to obtain

getElementsByTagName()

[JavaScriptMethodDomGetElementsByTagName]

The `getElementsByTagName()` method (when applied to the document object of the JavaScript DOM) returns a NodeList containing all the elements with the specified tag name (i.e. element type).

It has the following syntax with the following parameters. It returns a NodeList representing a collection of all relevant elements, ordered as they appear in the source code.

```
document.getElementsByTagName(tagname)
```

Parameter	Required / Optional	Description
tagame	Required	String specifying the tag name of the elements you want to obtain

hasFocus()

[JavaScriptMethodDOMHasFocus]

The `hasFocus()` method (when applied to the document object of the JavaScript DOM) returns `true` if document has focus, otherwise returns `false`.

It has the following syntax with no parameters. It returns a Boolean value indicating whether the document (or any element within it) has focus (loosely speaking where the cursor currently is).

```
document.hasFocus()
```

importNode()

[JavaScriptMethodDOMImportNode]

The `importNode()` method (when applied to the document object of the JavaScript DOM) imports node from another document (i.e. creates a copy of and inserts the copy into the current document).

It has the following syntax with the following parameters. It returns a Node object representing the imported node.

`document.importNode(`*node, deepness*`)`

Parameter	Required / Optional	Description
node	Required	Node object from other document (any node type is permitted)
deepness	Required	Boolean value, which if `false` means only import the node itself, but if `true` then also import all child nodes, i.e. descendents, of the node

Note: if you want to copy a node from the current document then use `element.cloneNode()` and if you want to remove a node from one document an import it into another than use `document.adoptNode()`.

normalize()

[JavaScriptMethodDomNormalize]

The `normalize()` method (when applied to the document object of the JavaScript DOM) removes empty text nodes and joins adjacent text nodes.

It has the following syntax with no parameters. It does not return a value.

```
document.normalize()
```

It can also be applied to any node within the document using e.g. *node*`.normalize()`.

normalizeDocument()

[JavaScriptMethodDomNormalizeDocument]

In theory, the `normalizeDocument()` method (when applied to the document object of the JavaScript DOM) removes empty text nodes and joins adjacent text nodes. It has the same effect (when applied to the document as a whole) as `normalize()` (but the latter can also be applied at lower node levels in isolation).

In practice, `normalizeDocument()` does not currently seem to be supported by major browsers, so it is likely to be more robust to use `document.normalize()` instead.

It has the following syntax with no parameters. It does not return a value.

```
document.normalizeDocument()
```

open()

[JavaScriptMethodDomOpen]

The `open()` method (when applied to the document object of the JavaScript DOM) opens an HTML output stream (into which output from `write()` or `writeln()` can go).

It has the following syntax with the following parameters. It does not return any value.

```
document.open(MIMEtype, replace)
```

Parameter	Required / Optional	Description
MIMEtype	Optional	The MIME type of the document being written to (default is `"text/html"`.
`replace`	Optional	If set (i.e. present) then the history entry for the new document inherits the one for the document which opened it

Once all writes to the document have taken place, the document.close() method causes any output to be displayed. If a document already exists in the target then it will be cleared.

querySelector()

[JavaScriptMethodDomQuerySelector]

The `querySelector()` method (when applied to the document object of the JavaScript DOM) returns first (child) element that matches specified CSSSelector.

It has the following syntax with the following parameters. It returns the object representing the first element that matches the specified CSSSelector, or null if no matches are found. If the selector(s) is invalid then it throws a SYNTAX_ERR exception.

```
document.querySelector(CSSSelectors)
```

Parameter	Required / Optional	Description
CSSSelectors	Required	String specifying one or more CSSSelectors. For multiple selectors, separate each one with a comma.

querySelectorAll()

[JavaScriptMethodDomQuerySelectorAll]

The `querySelectorAll()` method (when applied to the document object of the JavaScript DOM) returns first (child) element that matches specified CSSSelector.

It has the following syntax with the following parameters. It returns a NodeList object representing the first element that matches the specified CSSSelector or if no such element exists it does not return any value. If the selector(s) is invalid then it throws a SYNTAX_ERR exception. The number of such elements can be identified from the length property of the NodeList object, and individual elements can then be accessed using relevant index numbers applied to the NodeList object.

`document.querySelectorAll(`*CSSSelectors*`)`

Parameter	Required / Optional	Description
CSSSelectors	Required	String specifying one or more CSSSelectors. For multiple selectors, separate each one with a comma.

removeEventListener()

[JavaScriptMethodDomRemoveEventListener]

The `removeEventListener()` method (when applied to the document object of the JavaScript DOM) removes (detaches) an event handler to the document.

It has the following syntax with the following parameters. It does not return any value.

`document.removeEventListener(`*event*`, `*function*`, `*useCapture*`)`

Parameter	Required / Optional	Description
event	Required	String specifying event to remove (excluding the '`on`' part at the start of the relevant event attribute name)
function	Required	Name of function (without '`()`' included at end)
useCapture	Optional	If `true` then event handler is removed from the capturing phase of the page load, if false then removed from the bubbling phase

Some earlier versions of some major browsers do not support this method. For these browsers you instead need to use the `detachEvent()` method. If the event listener was attached two times, once in the capturing and ones in the bubbling phase using the *useCapture* parameter then it needs to be removed twice as well.

renameNode()

[JavaScriptMethodDomRenameNode]

In theory, the `renameNode()` method (when applied to the document object of the JavaScript DOM) renames the specified node.

In practice, `renameNode()` does not currently seem to be supported by major browsers, so it is likely to be more robust to create the new node as desired, add it to the document in the appropriate place and then delete the old node.

It has the following syntax with the following parameters. It returns a Node object.

`document.renameNode(`*node, namespaceURI, newtagname*`)`

Parameter	Required / Optional	Description
node	Required	Node object representing node to be renamed (i.e. given a new node type)
namespaceURI	Required	Namespace URI of node, but can be set to `null` if you don't want to specify it
newnodename	Required	New tag name

write()

[JavaScriptMethodDomWrite]

The `write()` method (when applied to the document object of the JavaScript DOM) writes HTML (which can include JavaScript code) to the document.

It is mostly used for testing, as if it is used after an HTML document is fully loaded it will delete the existing HTML. Alternatively, it may be used to write to a bespoke output stream opened by the `document.open()` method.

It has the following syntax with the following parameters. It does not return any value.

`document.write(`*expr1, expr2, expr3, …*`)`

Parameter	Required / Optional	Description
expr1, expr2, expr3, …	Optional	Text written to output stream

writeln()

[JavaScriptMethodDomWriteln]

The `writeln()` method (when applied to the document object of the JavaScript DOM) writes HTML (which can include JavaScript code) to the document, writing a newline character after each expression.

It is mostly used for testing, as if it is used after an HTML document is fully loaded it will delete the existing HTML. Alternatively, it may be used to write to a bespoke output stream opened by the `document.open()` method.

It is essentially the same as `document.write()` except that it adds a new line character after each statement.

It has the following syntax with the following parameters. It does not return any value.

`document.writeln(`*`expr1, expr2, expr3, …`*`)`

Parameter	Required / Optional	Description
expr1, expr2, expr3, …	Optional	Text written to output stream

2. Properties and Methods for HTML Elements

[JavaScriptTutorialDOMDetails2]

HTML elements within the JavaScript DOM support the following properties and methods:

Properties:

Property	Description	More
`accessKey`	Sets/returns the accesskey attribute	Here
`attributes`	Returns NamedNodeMap of attributes	Here
`childElementCount`	Returns number of child elements	Here
`childNodes`	Returns collection of the child elements (including text and comment nodes)	Here
`children`	Returns collection of the child elements (excluding text and comment nodes)	Here
`classList`	Returns the class name(s) of an element	Here
`className`	Sets / returns the class attribute	Here
`clientHeight`	Returns height, including padding	Here
`clientLeft`	Returns width of left border	Here
`clientTop`	Returns width of top border	Here
`clientWidth`	Returns width, including padding	Here
`contentEditable`	Sets / returns whether content is editable	Here
`dir`	Sets / returns the dir attribute	Here
`firstChild`	Returns first child node	Here
`firstElementChild`	Returns first child element	Here
`id`	Sets / returns the id attribute	Here
`innerHTML`	Sets / returns the element's content	Here
`isContentEditable`	Returns `true` if element contents are editable, otherwise returns `false`	Here
`lang`	Sets / returns the lang attribute	Here
`lastChild`	Returns last child node	Here
`lastElementChild`	Returns last child element	Here
`length`	Returns number of nodes in NodeList	Here
`namespaceURI`	Returns namespace URI of element	Here
`nextElementSibling`	Returns next element at same node tree level	Here
`nextSibling`	Returns next node at same node tree level	Here
`nodeName`	Returns node name	Here
`nodeType`	Returns node type	Here
`nodeValue`	Sets/returns node value	Here
`offsetHeight`	Returns element height, including padding, border and scroll bar	Here
`offsetWidth`	Returns element width, including padding,	Here

	border and scroll bar	
`offsetLeft`	Returns horizontal offset position	Here
`offsetParent`	Returns offset container of element	Here
`offsetTop`	Returns vertical offset position	Here
`ownerDocument`	Returns root element (i.e. document object) within which element resides	Here
`parentNode`	Returns parent node	Here
`previousElementSibling`	Returns previous element at same node tree level	Here
`previousSibling`	Returns previous node at same node tree level	Here
`scrollHeight`	Returns entire height of element	Here
`scrollLeft`	Sets / returns number of pixels content is scrolled horizontally	Here
`scrollTop`	Sets / returns number of pixels content is scrolled vertically	Here
`scrollWidth`	Returns entire width of element	Here
`style`	Sets / returns the style attribute	Here
`tabIndex`	Sets / returns the tabindex attribute	Here
`tagName`	Returns the tag name of element (i.e. its type of element)	Here
`textContent`	Sets / returns the text content of a node and its descendants	Here
`title`	Sets / returns the title attribute	Here

Methods:

Method	Description	More
`addEventListener()`	Attaches an event handler	Here
`appendChild()`	Adds a new child after last existing one	Here
`blur()`	Removes focus from element	Here
`click()`	Simulates a mouse click on element	Here
`cloneNode()`	Clones an element	Here
`compareDocumentPosition()`	Compares position in document of two elements	Here
`contains()`	Returns `true` if node is a descendant of other node, otherwise returns `false`	Here
`focus()`	Gives focus to element	Here
`getAttribute()`	Returns specified attribute value	Here
`getAttributeNode()`	Returns specified attribute node	Here
`getElementsByClassName()`	Returns NodeList containing all elements with specified class attribute	Here
`getElementsByTagName()`	Returns NodeList containing all elements with specified tag name (i.e. element type)	Here
`hasAttribute()`	Returns `true` if element has specified attribute, otherwise returns `false`	Here
`hasAttributes()`	Returns `true` if element has any attributes, otherwise returns `false`	Here
`hasChildNodes()`	Returns `true` if element has any child nodes, otherwise returns `false`	Here
`insertBefore()`	Inserts new child node before specific existing	Here

	child node	
`isDefaultNamespace()`	Returns `true` if a specified namespace URI is the default namespace, otherwise returns `false`	Here
`isEqualNode()`	Returns `true` if two elements / nodes are 'equal' (but not necessarily exactly the same), otherwise returns `false`	Here
`isSameNode()`	Returns `true` if two elements / nodes are the same (i.e. equal but also computationally refer to the same node), otherwise returns `false`	Here
`isSupported()`	Returns `true` if specified feature is supported, otherwise returns `false`	Here
`normalize()`	Removes empty text nodes and joins adjacent notes	Here
`querySelector()`	Returns first (child) element that matches specified CSSSelector	Here
`querySelectorAll()`	Returns (NodeList) collection containing all (child) elements that match specified CSSSelector(s)	Here
`removeAttribute()`	Removes specified attribute	Here
`removeAttributeNode()`	Removes specified attribute node, and returns the removed node	Here
`removeChild()`	Removes specified child node	Here
`removeEventListener()`	Detaches (removes)an event handler	Here
`replaceChild()`	Replaces specified child node	Here
`scrollIntoView()`	Scroll specified element into visible area of browser window	Here
`setAttribute()`	Sets specified attribute	Here
`setAttributeNode()`	Sets specified attribute node	Here
`toString()`	Converts element to string	Here
`item()`	Returns node at specified index position in a NodeList	Here

Further comments:

Collections of nodes form NodeLists, so there is always a NodeList associated with an element (although it might be empty initially).

JavaScript DOM HTML properties:

accessKey

[JavaScriptPropertyDomHtmlAccessKey]

The `accessKey` property of HTML elements within the JavaScript DOM sets / returns its accesskey attribute.

attributes

[JavaScriptPropertyDomHtmlAttributes]

The `attributes` property of HTML elements within the JavaScript DOM returns the NamedNodeMap of its attributes.

childElementCount

[JavaScriptPropertyDomHtmlChildElementCount]

The `childElementCount` property of HTML elements within the JavaScript DOM returns the number of its child elements.

childNodes

[JavaScriptPropertyDomHtmlChildNodes]

The `childNodes` property of HTML elements within the JavaScript DOM returns a collection of its child elements (including text and comment nodes).

children

[JavaScriptPropertyDomHtmlChildren]

The `children` property of HTML elements within the JavaScript DOM returns a collection of its child elements (excluding text and comment nodes).

classList

[JavaScriptPropertyDomHtmlClassList]

The `classList` property of HTML elements within the JavaScript DOM returns the class name(s) of the element.

className

[JavaScriptPropertyDomHtmlClassName]

The `className` property of HTML elements within the JavaScript DOM sets / returns the class attribute of the element.

clientHeight

[JavaScriptPropertyDomHtmlClientHeight]

The `clientHeight` property of HTML elements within the JavaScript DOM returns the height (including padding) of the element.

clientLeft

[JavaScriptPropertyDomHtmlClientLeft]

The `clientLeft` property of HTML elements within the JavaScript DOM returns the width of left border of the element.

clientTop

[JavaScriptPropertyDomHtmlClientTop]

The `clientTop` property of HTML elements within the JavaScript DOM returns the width of top border of the element.

clientWidth

[JavaScriptPropertyDomHtmlClientWidth]

The `clientWidth` property of HTML elements within the JavaScript DOM returns the width (including padding) of the element.

contentEditable

[JavaScriptPropertyDomHtmlContentEditable]

The `contentEditable` property of HTML elements within the JavaScript DOM sets / returns whether the content of the element is editable.

dir

[JavaScriptPropertyDomHtmlDir]

The `dir` property of HTML elements within the JavaScript DOM sets / returns the dir attribute of the element.

firstChild

[JavaScriptPropertyDomHtmlFirstChild]

The `firstChild` property of HTML elements within the JavaScript DOM returns the first child node of the element.

firstElementChild

[JavaScriptPropertyDomHtmlFirstElementChild]

The `firstElementChild` property of HTML elements within the JavaScript DOM returns the first child element of the element.

id

[JavaScriptPropertyDomHtmlId]

The `id` property of HTML elements within the JavaScript DOM sets / returns the id attribute of the element.

innerHTML

[JavaScriptPropertyDomHtmlInnerHTML]

The `innerHTML` property of HTML elements within the JavaScript DOM sets / returns the element's content.

isContentEditable

[JavaScriptPropertyDomHtmlIsContentEditable]

The `isContentEditable` property of HTML elements within the JavaScript DOM returns `true` if the element's contents are editable, otherwise returns `false`.

lang

[JavaScriptPropertyDomHtmlLang]

The `lang` property of HTML elements within the JavaScript DOM sets / returns the lang attribute of the element.

lastChild

[JavaScriptPropertyDomHtmlLastChild, © Nematrian 2017]

The `lastChild` property of HTML elements within the JavaScript DOM returns the last child node of the element.

lastElementChild

[JavaScriptPropertyDomHtmlLastElementChild]

The `lastElementChild` property of HTML elements within the JavaScript DOM returns the last child element of the element.

namespaceURI

[JavaScriptPropertyDomHtmlNamespaceURI]

The `namespaceURI` property of HTML elements within the JavaScript DOM returns the namespace URI of the element.

nextElementSibling

[JavaScriptPropertyDomHtmlNextElementSibling]

The `nextElementSibling` property of HTML elements within the JavaScript DOM returns the next element at same node tree level of the element.

nextSibling

[JavaScriptPropertyDomHtmlNextSibling]

The `nextSibling` property of HTML elements within the JavaScript DOM returns the next node at same node tree level of the element.

nodeName

[JavaScriptPropertyDomHtmlNodeName]

The `nodeName` property of HTML elements within the JavaScript DOM returns the node name of the element.

nodeType

[JavaScriptPropertyDomHtmlNodeType]

The `nodeType` property of HTML elements within the JavaScript DOM returns the node type of the element. It is a number:

Type of node	nodeType property
Element node	1
Attribute node	2
Text node	3
Comment node	8

nodeValue

[JavaScriptPropertyDomHtmlNodeValue]

The `nodeValue` property of HTML elements within the JavaScript DOM sets / returns the node value of the element.

offsetHeight

[JavaScriptPropertyDomHtmlOffsetHeight]

The `offsetHeight` property of HTML elements within the JavaScript DOM returns the element height, including padding, border and scroll bar.

offsetLeft

[JavaScriptPropertyDomHtmlOffsetLeft]

The `offsetLeft` property of HTML elements within the JavaScript DOM returns the horizontal offset position of the element.

offsetParent

[JavaScriptPropertyDomHtmlOffsetParent]

The `offsetParent` property of HTML elements within the JavaScript DOM returns the offset container of the element.

offsetTop

[JavaScriptPropertyDomHtmlOffsetTop]

The `offsetTop` property of HTML elements within the JavaScript DOM returns the vertical offset position of the element.

offsetWidth

[JavaScriptPropertyDomHtmlOffsetWidth]

The `offsetWidth` property of HTML elements within the JavaScript DOM returns the element width, including padding, border and scroll bar.

ownerDocument

[JavaScriptPropertyDomHtmlOwnerDocument]

The `ownerDocument` property of HTML elements within the JavaScript DOM returns the root element (i.e. the document object) within which element resides.

parentNode

[JavaScriptPropertyDomHtmlParentNode]

The `parentNode` property of HTML elements within the JavaScript DOM returns the parent node of the element.

previousElementSibling

[JavaScriptPropertyDomHtmlPreviousElementSibling]

The `previousElementSibling` property of HTML elements within the JavaScript DOM returns the previous element at same node tree level of the element.

previousSibling

[JavaScriptPropertyDomHtmlPreviousSibling]

The `previousSibling` property of HTML elements within the JavaScript DOM returns the previous node at same node tree level of the element.

scrollHeight

[JavaScriptPropertyDomHtmlScrollHeight]

The `scrollHeight` property of HTML elements within the JavaScript DOM returns the entire height of the element.

scrollLeft

[JavaScriptPropertyDomHtmlScrollLeft]

The `scrollLeft` property of HTML elements within the JavaScript DOM sets / returns number of pixels that the content of the element is scrolled horizontally.

scrollTop

[JavaScriptPropertyDomHtmlScrollTop]

The `scrollTop` property of HTML elements within the JavaScript DOM sets / returns number of pixels that the content of the element is scrolled vertically.

scrollWidth

[JavaScriptPropertyDomHtmlScrollWidth]

The `scrollWidth` property of HTML elements within the JavaScript DOM returns the entire width of the element.

style

[JavaScriptPropertyDomHtmlStyle]

The `style` property of HTML elements within the JavaScript DOM sets / returns the style attribute of the element.

tabIndex

[JavaScriptPropertyDomHtmlTabIndex, © Nematrian 2017]

The `tabIndex` property of HTML elements within the JavaScript DOM sets / returns the tabindex attribute of the element.

tagName

[JavaScriptPropertyDomHtmlTagName]

The `tagName` property of HTML elements within the JavaScript DOM returns the tag name of the element (i.e. the type of element that it is).

textContent

[JavaScriptPropertyDomHtmlTextContent]

The `textContent` property of HTML elements within the JavaScript DOM returns the text content of a node and its descendants.

title

[JavaScriptPropertyDomHtmlTitle]

The `title` property of HTML elements within the JavaScript DOM sets / returns the title attribute of the element.

JavaScript DOM HTML methods

addEventListener()

[JavaScriptMethodDomHtmlAddEventListener]

The `addEventListener()` method (when applied to HTML elements in the JavaScript DOM) attaches an event handler to the element.

It has the following syntax with the following parameters. It does not return any value.

element`.addEventListener(`*event*`,` *function*`,` *useCapture*`)`

Parameter	Required / Optional	Description
event	Required	String specifying event (excluding the '`on`' part at the start of the relevant event attribute name)
function	Required	Name of function (with '`()`' included at end)
useCapture	Optional	If `true` then event handler is executed in the capturing phase of the page load, if false then in the bubbling phase

Some earlier versions of some major browsers do not support this method. For these browsers you instead need to use the `attachEvent()` method.

appendChild()

[JavaScriptMethodDomHtmlAppendChild]

The `appendChild()` method (when applied to HTML elements in the JavaScript DOM) adds a new child node after the last existing one of the element.

It has the following syntax with the following parameters. It returns a NodeList representing the added node.

element`.appendChild(`*node*`)`

Parameter	Required / Optional	Description
node	Required	The node object to be appended

blur()

[JavaScriptMethodDomHtmlBlur]

The `blur()` method (when applied to HTML elements in the JavaScript DOM) removes focus from the element.

It has the following syntax with no parameters. It does not return a value.

element`.blur()`

click()

[JavaScriptMethodDomHtmlClick]

The `click()` method (when applied to HTML elements in the JavaScript DOM) simulates a mouse click on element.

It has the following syntax with no parameters. It does not return a value.

element`.click()`

cloneNode()

[JavaScriptMethodDomHtmlCloneNode]

The `cloneNode()` method (when applied to HTML elements in the JavaScript DOM) Clones an element (i.e. creates a copy and returns that copy).

It has the following syntax with the following parameters. It returns a Node object representing the cloned node.

element`.cloneNode(`*deepness*`)`

Parameter	Required / Optional	Description
deepness	Required	Boolean value, which if `false` means only copy the node itself, but if `true` then also copy all child nodes, i.e. descendents, of the node

compareDocumentPosition()

[JavaScriptMethodDomHtmlCompareDocumentPosition]

The `compareDocumentPosition()` method (when applied to HTML elements in the JavaScript DOM) compares the position in the document of two elements. It involves the sum of the following:

Contribution	Meaning
1	No relationship: *n1* and *n2* don't belong to same document
2	*n1* after *n2*
4	*n1* before *n2*
8	*n1* inside *n2*
16	*n2* inside *n1*
32	No relationship or the two nodes are attribute nodes of the same element

where *n1* and *n2* would be nodes in the form: *n1*`.compareDocumentPosition(`*n2*`)`

It has the following syntax with the following parameters (when applied to elements). It returns a number.

element`.compareDocumentPosition(`*node*`)`

Parameter	Required / Optional	Description
node	Required	The node object to compare with the current node or element

contains()

[JavaScriptMethodDomHtmlContains]

The `contains()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if one node is a descendant of another node, otherwise returns `false`.

It has the following syntax with the following parameters (when applied to elements). It returns a Boolean which is `true` if `node` is a descendant of `element`.

element`.contains(`*node*`)`

Parameter	Required / Optional	Description
node	Required	The node object to compare with the current node or element

focus()

[JavaScriptMethodDomHtmlFocus]

The `focus()` method (when applied to HTML elements in the JavaScript DOM) gives focus to the element.

It has the following syntax with no parameters. It does not return a value.

element`.focus()`

getAttribute()

[JavaScriptMethodDomHtmlGetAttribute]

The `getAttribute()` method (when applied to HTML elements in the JavaScript DOM) returns the value of the specified attribute.

It has the following syntax with the following parameters. It returns a String representing the value of the specified attribute. If the attribute does not exist then the return value will be `null` or an empty string, i.e. `""`.

element`.getAttribute(`*attributename*`)`

Parameter	Required / Optional	Description
attributename	Required	String containing name of attribute

getAttributeNode()

[JavaScriptMethodDomHtmlGetAttributeNode]

The `getAttributeNode()` method (when applied to HTML elements in the JavaScript DOM) returns the specified attribute node.

It has the following syntax with the following parameters. It returns an Attr object representing the specified attribute. If the attribute does not exist then the return value will be `null` or an empty string, i.e. `""`.

element`.getAttributeNode(`*attributename*`)`

Parameter	Required / Optional	Description
attributename	Required	String containing name of attribute

getElementsByClassName()

[JavaScriptMethodDomHtmlGetElementsByClassName]

The `getElementsByClassName()` method (when applied HTML elements in the JavaScript DOM) returns a NodeList containing all the elements within the element with the specified class attribute.

It has the following syntax with the following parameters. It returns a NodeList representing a collection of all relevant elements, ordered as they appear in the source code.

element`.getElementsByClassName(`*classname*`)`

Parameter	Required / Optional	Description
classname	Required	String specifying the class name of the elements you want to obtain. To include multiple class names, separate individual class names by spaces

getElementsByTagName()

[JavaScriptMethodDomHtmlGetElementsByTagName]

The `getElementsByTagName()` method (when applied to HTML elements in the JavaScript DOM) returns a NodeList containing all the elements with the specified tag name (i.e. element type).

It has the following syntax with the following parameters. It returns a NodeList representing a collection of all relevant elements, ordered as they appear in the source code.

element`.getElementsByTagName(`*tagname*`)`

Parameter	Required / Optional	Description
tagame	Required	String specifying the tag name of the elements you want to obtain

hasAttribute()

[JavaScriptMethodDomHtmlHasAttribute]

The `hasAttribute()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if the element has the specified attribute, otherwise it returns `false`.

It has the following syntax with the following parameters. It returns a Boolean as above.

element`.hasAttribute(`*attributename*`)`

Parameter	Required / Optional	Description
attributename	Required	String containing name of attribute

hasAttributes()

[JavaScriptMethodDomHtmlHasAttributes]

The `hasAttributes()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if the element has an attributes, otherwise it returns `false`.

It has the following syntax with no parameters. It returns a Boolean as above. It can be applied to any node, but if the node is not an element then it will always return `false`.

element`.hasAttributes()`

hasChildNodes()

[JavaScriptMethodDomHtmlHasChildNodes]

The `hasChildNodes()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if the element has any child nodes, otherwise it returns `false`. Note: whitespace inside a node is deemed to form a (child) text node, and so will result in a `true` value being returned by this method.

It has the following syntax with no parameters. It returns a Boolean as above. It can be applied to any node, but if the node is not an element then it will always return `false`.

element`.hasChildNodes()`

insertBefore()

[JavaScriptMethodDomHtmlInsertBefore]

The `insertBefore()` method (when applied to HTML elements in the JavaScript DOM) inserts a new child node before an existing child node.

It has the following syntax with the following parameters. It returns a Node object representing the new child node.

element`.insertBefore(`*newnode, existingnode*`)`

Parameter	Required / Optional	Description
newnode	Required	The node object to be inserted
existingnode	Optional	The node object before which the new node is to be inserted. If not specified then the new node will be inserted at the end of the element

isDefaultNamespace()

[JavaScriptMethodDomHtmlIsDefaultNamespace]

The `isDefaultNamespace()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if a specified namespace URI is the default namespace, otherwise returns `false`.

It has the following syntax with the following parameters. It returns a Boolean as above.

element`.isDefaultNamespace(`*namespaceURI*`)`

Parameter	Required / Optional	Description
namespaceURI	Required	String corresponding to specified namespace URI, e.g. `"http://www.w3.org/1999/xhtml"`

isEqualNode()

[JavaScriptMethodDomHtmlIsEqualNode]

The `isEqualNode()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if two elements / nodes are 'equal', otherwise returns `false`. Two nodes are deemed 'equal' if all the following are true, namely that they have the same:

- Node type
- nodeName, nodeValue, localName, namespaceURI and prefix
- childNodes (including all descendants)

- same attributes and attribute values (although the attributes do not need to be in the same order)

It has the following syntax with the following parameters. It returns a Boolean as above.

element`.isEqualNode(`*node*`)`

Parameter	Required / Optional	Description
node	Required	Node object which is compared to the element (method can also be applied to a node object)

isSameNode()

[JavaScriptMethodDomHtmlIsSameNode]

The `isSameNode()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if two elements / nodes are the same (i.e. not just equal but also computationally refer to the same node), otherwise returns `false`.

Some major browsers no longer support this method, so it is more robust to use the identically equal operator ===.

It has the following syntax with the following parameters. It returns a Boolean as above.

element`.isSameNode(`*node*`)`

Parameter	Required / Optional	Description
node	Required	Node object which is compared to the element (method can also be applied to a node object)

isSupported()

[JavaScriptMethodDomHtmlIsSupported]

The `isSupported()` method (when applied to HTML elements in the JavaScript DOM) returns `true` if specified feature is supported, otherwise returns `false`.

Some major browsers no longer support this method, and so it is desirable not to use it.

It has the following syntax with the following parameters. It returns a Boolean as above.

element`.isSupported(`*feature, version*`)`

Parameter	Required / Optional	Description
feature	Required	String defining feature being tested
version	Option	String defining version of feature being tested

item()

[JavaScriptMethodDomHtmlItem]

The `item()` method (when applied to HTML elements in the JavaScript DOM) returns the node at the specified index position in a NodeList.

It has the following syntax with the following parameters. It returns a node object representing the node at the relevant index value, or `null` if the index is outside the applicable range.

nodelist`.item(`*index*`)` or *nodelist*`[`*index*`]`

Parameter	Required / Optional	Description
index	Required	Number representing the index of node to be returned (the index is zero-based, i.e. starts at zero)

normalize()

[JavaScriptMethodDomHtmlNormalize]

The `normalize()` method (when applied to HTML elements in the JavaScript DOM) removes empty text nodes and joins adjacent text nodes.

It has the following syntax with no parameters. It does not return a value.

element`.normalize()`

querySelector()

[JavaScriptMethodDomHtmlQuerySelector]

The `querySelector()` method (when applied to HTML elements in the JavaScript DOM) returns first (child) element that matches specified CSSSelector.

It has the following syntax with the following parameters. It returns the object representing the first element that matches the specified CSSSelector, or null if no matches are found. If the selector(s) is invalid then it throws a SYNTAX_ERR exception.

element`.querySelector(`*CSSSelectors*`)`

Parameter	Required / Optional	Description
CSSSelectors	Required	String specifying one or more CSSSelectors. For multiple selectors, separate each one with a comma.

querySelectorAll()

[JavaScriptMethodDomHtmlQuerySelectorAll]

The `querySelectorAll()` method (when applied to HTML elements in the JavaScript DOM) returns first (child) element that matches specified CSSSelector.

It has the following syntax with the following parameters. It returns a NodeList object representing the first element that matches the specified CSSSelectoroes not return any value. If the selector(s) is

invalid then it throws a SYNTAX_ERR exception. The number of such elements can be identified from the length property of the NodeList object, and individual elements can then be accessed using relevant index numbers applied to the NodeList object.

element`.querySelectorAll(`*CSSSelectors*`)`

Parameter	Required / Optional	Description
CSSSelectors	Required	String specifying one or more CSSSelectors. For multiple selectors, separate each one with a comma.

removeAttribute()

[JavaScriptMethodDomHtmlRemoveAttribute]

The `removeAttribute()` method (when applied to HTML elements in the JavaScript DOM) removes the specified attribute.

It has the following syntax with the following parameters. It does not return a value.

element`.removeAttribute(`*attributename*`)`

Parameter	Required / Optional	Description
attributename	Required	String defining name of attribute to be removed

removeAttributeNode()

[JavaScriptMethodDomHtmlRemoveAttributeNode]

The `removeAttributeNde()` method (when applied to HTML elements in the JavaScript DOM) removes the specified attribute.

It has the following syntax with the following parameters. It returns an Attr object representing the attribute node that has been removed.

element`.removeAttributeNode(`*attributenode*`)`

Parameter	Required / Optional	Description
attributenode	Required	An Attr object being the attribute node to be removed

removeChild()

[JavaScriptMethodDomHtmlRemoveChild]

The `removeChild()` method (when applied to HTML elements in the JavaScript DOM) removes the specified child node.

It has the following syntax with the following parameters. It returns the removed node (or `null` if the node does not exist).

element`.removeChild(`*node*`)`

Parameter	Required / Optional	Description
node	Required	Node object to be removed

removeEventListener()

[JavaScriptMethodDomHtmlRemoveEventListener]

The `removeEventListener()` method (when applied to HTML elements in the JavaScript DOM) removes (detaches) an event handler from the element.

It has the following syntax with the following parameters. It does not return any value.

element`.removeEventListener(`*event*`,` *function*`,` *useCapture*`)`

Parameter	Required / Optional	Description
event	Required	String specifying event (excluding the '`on`' part at the start of the relevant event attribute name)
function	Required	Name of function (with '`()`' included at end)
useCapture	Optional	If `true` then event handler is executed in the capturing phase of the page load, if false then in the bubbling phase

Some earlier versions of some major browsers do not support this method. For these browsers you instead need to use the `detachEvent()` method. If the event listener was attached two times, once in the capturing and ones in the bubbling phase using the *useCapture* parameter then it needs to be removed twice as well.

replaceChild()

[JavaScriptMethodDomHtmlReplaceChild]

The `replaceChild()` method (when applied to HTML elements in the JavaScript DOM) replaces the specified child node.

It has the following syntax with the following parameters. It returns the replaced node (or `null` if the node does not exist).

element`.replaceChild(`*newnode, oldnode*`)`

Parameter	Required / Optional	Description
newnode	Required	Node object to be inserted
oldnode	Required	Node object to be removed

scrollIntoView()

[JavaScriptMethodDomHtmlScrollIntoView]

The `scrollIntoView()` method (when applied to HTML elements in the JavaScript DOM) scrolls specified element into visible area of browser window.

It has the following syntax with the following parameters. It does not return a value.

element`.scrollIntoView(`*alignment*`)`

Parameter	Required / Optional	Description
alignment	Optional	Boolean. If `true` the top of element will be aligned with the top of the visible area of the scrollable ancestor, if `false` then the bottoms aligned instead. If omitted then tops are aligned

setAttribute()

[JavaScriptMethodDomHtmlSetAttribute]

The `setAttribute()` method (when applied to HTML elements in the JavaScript DOM) sets the specified attribute value.

It has the following syntax with the following parameters. It does not return a value.

element`.setAttribute(`*attributename, attributevalue*`)`

Parameter	Required / Optional	Description
attributename	Required	String indicating name of attribute to be set
attributevalue	Required	String indicating value of that attribute

setAttributeNode ()

[JavaScriptMethodDomHtmlSetAttributeNode]

The `setAttributeNode()` method (when applied to HTML elements in the JavaScript DOM) sets specified attribute node (i.e. adds it to the element if it didn't previously exist, or replaces the old one, if the element already had the attribute).

It has the following syntax with the following parameters. It does not return a value.

element`.setAttributeNode(`*attributenode*`)`

Parameter	Required / Optional	Description
attributenode	Required	An Attr object representing the attribute node to be set

toString()

[JavaScriptMethodDomHtmlToString]

The `toString()` method (when applied to HTML elements in the JavaScript DOM) converts the element to string.

It has the following syntax with no parameters. It returns a String value as above.

element`.toString()`

3. Properties and Methods for HTML Attributes

[JavaScriptTutorialDOMDetails3]

HTML attributes (Attr objects) within the JavaScript DOM support the following generic properties:

Properties:

Property	Description	More
`isId`	Returns `true` if attribute is of type ID, otherwise returns `false`	Here
`name`	Returns name of attribute	Here
`value`	Sets/returns value of attribute	Here
`specified`	Returns `true` if attribute has been specified, otherwise returns `false`	Here

Further comments:

In the W3C DOM Core, the Attr object inherits all properties and methods from the Node object. However, many of these properties and methods aren't meaningful for HTML attributes. Also, in DOM 4 this inheritance will no longer apply, so it is desirable not to use node object properties and methods on Attr objects.

JavaScript DOM HTML attribute properties:

isId

[JavaScriptPropertyAttrIsId]

The `isId` property of HTML attribute objects within the JavaScript DOM returns `true` if the attribute is an ID attribute, otherwise returns `false`.

Most browsers no longer seem to support this property.

name

[JavaScriptPropertyAttrName]

The `name` property of HTML attribute objects within the JavaScript DOM returns the name of the attribute.

specified

[JavaScriptPropertyAttrSpecified]

The `specified` property of HTML attribute objects within the JavaScript DOM returns `true` if the attribute has been specified, otherwise returns `false`.

value

[JavaScriptPropertyAttrValue]

The `value` property of HTML attribute objects within the JavaScript DOM sets / returns the value of the attribute.

4. Properties and Methods for NamedNodeMap objects

[JavaScriptTutorialDOMDetails4]

NamedNodeMap objects within the JavaScript DOM support the following properties and methods:

Properties:

Property	Description	More
`length`	Returns number of nodes in the NamedNodeMap	Here

Methods:

Method	Description	More
`getNamedItem()`	Returns specified node from NamedNodeMap, specified by its name	Here
`item()`	Returns node at specified position from NamedNodeMap	Here
`removeNamedItem()`	Removes specified node	Here
`setNamedItem()`	Adds / sets specified node (by name)	Here

JavaScript DOM NamedNodMap properties:

length

[JavaScriptPropertyNamedNodeMapLength]

The `length` property of NamedNodeMap objects within the JavaScript DOM returns number of nodes in the NamedNodeMap.

JavaScript DOM NamedNodeMap methods:

getNamedItem()

[JavaScriptMethodNamedNodeMapGetNamedItem]

The `getNamedItem()` method (when applied to NamedNodeMap objects in the JavaScript DOM) returns a specified node from a NamedNodeMap, specified by its name.

It has the following syntax with the following parameters. It returns a node object representing the node at the relevant index value, or `null` if the index is outside the applicable range.

namednodemap`.getNamedItem(`*name*`)`

Parameter	Required / Optional	Description
name	Required	String containing name of node to be returned

item()

[JavaScriptMethodNamedNodeMapItem]

The `item()` method (when applied to NamedNodeMap objects in the JavaScript DOM) returns the node at the specified index position in a NamedNodeMap.

It has the following syntax with the following parameters. It returns a node object representing the node at the relevant index value, or `null` if the index is outside the applicable range.

namednodemap`.item(`*index*`)` or *namednodemap*`[`*index*`]`

Parameter	Required / Optional	Description
index	Required	Number representing the index of node to be returned (the index is zero-based, i.e. starts at zero)

removeNamedItem()

[JavaScriptMethodNamedNodeMapRemoveNamedItem]

The `removeNamedItem()` method (when applied to NamedNodeMap objects in the JavaScript DOM) removes a specified node, specified by its name.

It has the following syntax with the following parameters. It returns a node object representing the removed node.

namednodemap`.removeNamedItem(`*name*`)`

Parameter	Required / Optional	Description
name	Required	String containing name of node to be removed

setNamedItem()

[JavaScriptMethodNamedNodeMapSetNamedItem]

The `setNamedItem()` method (when applied to NamedNodeMap objects in the JavaScript DOM) adds / sets a specified node (specified by name). If the node already existed then the old node will be replaced. If it didn't previously exist then it will be addNote: if you are setting an element attribute then you can use the *element*.setAttribute() instead.

It has the following syntax with the following parameters. It returns a node object representing the replaced object (if any), or `null` if no replacement occurred.

namednodemap`.setNamedItem(`*node*`)`

Parameter	Required / Optional	Description
node	Required	Node object to be added or to replace the relevant old node

5. Properties and Methods for Event objects

[JavaScriptTutorialDOMDetails5]

Event objects within the JavaScript DOM support the following constants, properties and methods:

Constants:

Property	Description	More
`AT_TARGET`	Event is in the target phase, i.e. being evaluated at the event target	Here
`BUBBLING_PHASE`	Event is in the bubbling phase	Here
`CAPTURING_PHASE`	Event is in the capture phase	Here

Properties:

Property	Description	More
`bubbles`	Returns `true` if event is a bubbling event, otherwise returns `false`	Here
`cancelable`	Returns `true` if event can have its default action prevented (i.e. cancelled), otherwise returns `false`	Here
`currentTarget`	Returns element whose event listener(s) triggered event	Here
`defaultPrevented`	Returns `true` if `preventDefault()` method was called for event, otherwise returns `false`	Here
`eventPhase`	Returns which phase of event flow is currently being evaluated	Here
`isTrusted`	Returns `true` if event is trusted, otherwise returns `false`	Here
`target`	Returns element that triggered event	Here
`timeStamp`	Returns time at which event was created	Here
`type`	Returns name of event	Here
`view`	Returns reference to Window object where event occurred	Here

Methods:

Method	Description	More
`preventDefault()`	Cancels event if it is cancellable (i.e. default action belonging to event will not occur)	Here
`stopImmediatePropagation()`	Prevents other event listeners of the same event from being called	Here
`stopPropagation()`	Prevents further propagation of an event	Here

JavaScript DOM Event constants:

AT_TARGET

[JavaScriptPropertyEventAT_TARGET]

The `AT_TARGET` constant / property of Event objects within the JavaScript DOM indicates whether the event is in the target phase, i.e. being evaluated at the event target.

BUBBLING_PHASE

[JavaScriptPropertyEventBUBBLING_PHASE]

The `BUBBLING_PHASE` constant / property of Event objects within the JavaScript DOM indicates whether the event is in the bubbling phase.

CAPTURING_PHASE

[JavaScriptPropertyEventCAPTURING_PHASE]

The `CAPTURING_PHASE` constant / property of Event objects within the JavaScript DOM indicates whether the event is in the capture phase.

JavaScript DOM Event properties:

bubbles

[JavaScriptPropertyEventBubbles]

The `bubbles` property of Event objects within the JavaScript DOM returns `true` if the event is a bubbling event, otherwise returns `false`.

cancelable

[JavaScriptPropertyEventCancelable]

The `cancelable` property of Event objects within the JavaScript DOM returns `true` if event can have its default action prevented (i.e. cancelled), otherwise returns `false`.

currentTarget

[JavaScriptPropertyEventCurrentTarget]

The `currentTarget` property of Event objects within the JavaScript DOM returns the element whose event listener(s) triggered the event.

defaultPrevented

[JavaScriptPropertyEventDefaultPrevented]

The `defaultPrevented` property of Event objects within the JavaScript DOM returns `true` if the `preventDefault()` method was called for the event, otherwise returns `false`.

eventPhase

[JavaScriptPropertyEventEventPhase]

The `eventPhase` property of Event objects within the JavaScript DOM returns which phase of event flow is currently being evaluated for the event.

isTrusted

[JavaScriptPropertyEventIsTrusted]

The `isTrusted` property of Event objects within the JavaScript DOM returns `true` if the event is trusted, otherwise returns `false`.

target

[JavaScriptPropertyEventTarget]

The `target` property of Event objects within the JavaScript DOM returns the HTML element that triggered event.

timeStamp

[JavaScriptPropertyEventTimeStamp]

The `timeStamp` property of Event objects within the JavaScript DOM returns the time at which event was created.

type

[JavaScriptPropertyEventType]

The `type` property of Event objects within the JavaScript DOM returns the name of the event.

view

[JavaScriptPropertyEventView]

The `view` property of Event objects within the JavaScript DOM returns a reference to the window object where the event occurred.

JavaScript DOM Event methods:

preventDefault()

[JavaScriptMethodEventPreventDefault]

The `preventDefault()` method (when applied to Event objects in the JavaScript DOM) cancels an event if it is cancellable (i.e. it causes the default action belonging to the event not to occur).

For example, this could stop the browser from going to a new page when a link is clicked.

Note: Not all events are cancellable; the *event*.cancelable property will indicate whether it is cancellable. Also, the `preventDefault()` method does not prevent further propagation through the DOM; to limit this, use the *event*.stopImmediatePropagation() or *event*.stopPropagation() methods.

It has the following syntax with no parameters. It does not return a value.

event`.preventDefault()`

stopImmediatePropagation()

[JavaScriptMethodEventStopImmediatePropagation]

The `stopImmediatePropagation()` method (when applied to Event objects in the JavaScript DOM) prevents other (later) event listeners of the same event from being called (so if we add several event listeners to the same event then they execute in turn but only up to the one containing this method).

It has the following syntax with no parameters. It does not return a value.

event`.stopImmediatePropagation()`

stopPropagation()

[JavaScriptMethodEventStopPropagation]

The `stopPropagation()` method (when applied to Event objects in the JavaScript DOM) prevents further propagation of the event in the capturing and bubbling phases of an event.

Note: 'bubbling' triggers additional event listeners (if appropriately defined) found by following the event target's parent chain upwards (up to and including the overall document), see W3C specifications for more details.

It has the following syntax with no parameters. It does not return a value.

event`.stopPropagation()`

6. Properties and Methods for MouseEvent objects

[JavaScriptTutorialDOMDetails6]

MouseEvent objects within the JavaScript DOM support the following properties:

Properties (when mouse event is triggered):

Property	Description	More
`altKey`	Returns `true` if 'ALT' key was pressed, otherwise returns `false`	Here
`button`	Returns which mouse button was pressed	Here
`buttons`	Returns which mouse buttons were pressed	Here
`clientX`	Returns horizontal coordinate of mouse pointer (relative to current window)	Here
`clientY`	Returns vertical coordinate of mouse pointer (relative to current window)	Here
`ctrlKey`	Returns `true` if 'CTRL' key was pressed, otherwise returns `false`	Here
`detail`	Returns number indicating number of times mouse was clicked	Here
`metaKey`	Returns `true` if 'META' key was pressed, otherwise returns `false`	Here
`pageX`	Returns horizontal coordinate of mouse pointer (relative to page)	Here
`pageY`	Returns vertical coordinate of mouse pointer (relative to page)	Here
`relatedTarget`	Returns element related to element that triggered mouse event	Here
`screenX`	Returns horizontal coordinate of mouse pointer (relative to screen)	Here
`screenY`	Returns vertical coordinate of mouse pointer (relative to screen)	Here
`shiftKey`	Returns `true` if 'SHIFT' key was pressed, otherwise returns `false`	Here
`which`	Returns which mouse button was pressed (more consistent between browsers than the `button` property)	Here

JavaScript DOM MouseEvent properties:

altKey

[JavaScriptPropertyMouseEventAltKey]

The `altKey` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns `true` if the 'ALT' key was pressed, otherwise returns `false`.

button

[JavaScriptPropertyMouseEventButton]

The `button` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns which mouse button was pressed.

buttons

[JavaScriptPropertyMouseEventButtons]

The `buttons` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns which mouse buttons were pressed.

clientX

[JavaScriptPropertyMouseEventClientX]

The `clientX` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the horizontal coordinate of the mouse pointer (relative to the current window).

clientY

[JavaScriptPropertyMouseEventClientY]

The `clientY` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the vertical coordinate of the mouse pointer (relative to the current window).

ctrlKey

[JavaScriptPropertyMouseEventCtrlKey]

The `ctrlKey` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns `true` if the 'CTRL' key was pressed, otherwise returns `false`.

detail

[JavaScriptPropertyMouseEventDetail]

The `detail` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the number of times the mouse was clicked.

metaKey

[JavaScriptPropertyMouseEventMetaKey]

The `metaKey` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns `true` if the 'META' key was pressed, otherwise returns `false`.

pageX

[JavaScriptPropertyMouseEventPageX]

The `pageX` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the horizontal coordinate of the mouse pointer (relative to the page / document).

pageY

[JavaScriptPropertyMouseEventPageY]

The `pageY` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the vertical coordinate of the mouse pointer (relative to the page / document).

relatedTarget

[JavaScriptPropertyMouseEventRelatedTarget]

The `relatedTarget` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the element related to the element that triggered the mouse event.

screenX

[JavaScriptPropertyMouseEventScreenX]

The `screenX` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the horizontal coordinate of the mouse pointer (relative to the screen).

screenY

[JavaScriptPropertyMouseEventScreenY]

The `screenY` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns the vertical coordinate of the mouse pointer (relative to the screen).

shiftKey

[JavaScriptPropertyMouseEventShiftKey]

The `shiftKey` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns `true` if the 'SHIFT' key was pressed, otherwise returns `false`.

which

[JavaScriptPropertyMouseEventWhich]

The `which` property of MouseEvent objects within the JavaScript DOM (i.e. events triggered by an action with the mouse) returns which mouse button was pressed (its output is more consistent between browsers than the button property).

7. Properties and Methods for KeyboardEvent objects

[JavaScriptTutorialDOMDetails7]

KeyboardEvent objects within the JavaScript DOM support the following properties:

Properties (when keyevent is triggered):

Property	Description	More
`altKey`	Returns `true` if 'ALT' key was pressed, otherwise returns `false`	Here
`ctrlKey`	Returns `true` if 'CTRL' key was pressed, otherwise returns `false`	Here
`charCode`	Returns Unicode character code of key triggering onkeypress event	Here
`key`	Returns key value of key represented by event	Here
`keyCode`	(Included for compatibility only, latest specification recommends using `key` property). Returns Unicode character code of key pressed (for onkeypress event), or Unicode key code of key that triggered onkeydown or onkeyup event	Here
`location`	Returns location of key on keyboard or device	Here
`metaKey`	Returns `true` if 'META' key was pressed, otherwise returns `false`	Here
`shiftKey`	Returns `true` if 'SHIFT' key was pressed, otherwise returns `false`	Here
`which`	(Included for compatibility only, latest specification recommends using `key` property). Returns Unicode character code of key pressed (for onkeypress event), or Unicode key code of key that triggered onkeydown or onkeyup event	Here

JavaScript DOM KeyboardEvent properties:

altKey

[JavaScriptPropertyKeyboardEventAltKey]

The `altKey` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns `true` if the 'ALT' key was pressed, otherwise returns `false`.

charCode

[JavaScriptPropertyKeyboardEventCharCode]

The `charCode` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns the Unicode character code of the key triggering the onkeypress event.

ctrlKey

[JavaScriptPropertyKeyboardEventCtrlKey]

The `ctrlKey` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns `true` if the 'CTRL' key was pressed, otherwise returns `false`.

key

[JavaScriptPropertyKeyboardEventKey]

The `key` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns the key value of the key represented by the event.

keyCode

[JavaScriptPropertyKeyboardEventKeyCode]

The `keyCode` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns the Unicode character code of the key pressed (for an onkeypress event), or the Unicode key code of the key that triggered an onkeydown or an onkeyup event. It is included for compatibility only, as the latest specification recommends using the key property.

location

[JavaScriptPropertyKeyboardEventLocation]

The `location` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns the location of the key on the keyboard or device.

metaKey

[JavaScriptPropertyKeyboardEventMetaKey]

The `metaKey` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns `true` if the 'META' key was pressed, otherwise returns `false`.

shiftKey

[JavaScriptPropertyKeyboardEventShiftKey]

The `metaKey` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns `true` if the 'META' key was pressed, otherwise returns `false`.

which

[JavaScriptPropertyKeyboardEventWhich]

The `which` property of KeyboardEvent objects within the JavaScript DOM (i.e. events triggered by an action with the keyboard) returns the Unicode character code of the key pressed (for an onkeypress event), or the Unicode key code of the key that triggered an onkeydown or an onkeyup event. It is included for compatibility only, as the latest JavaScript specification recommends using the key property.

8. Properties and Methods for HashChangeEvent objects

[JavaScriptTutorialDOMDetails8]

HashChangeEvent objects within the JavaScript DOM support the following properties:

Properties (when keyevent is triggered):

Property	Description	More
`newURL`	Returns the URL of the document after the hash has been changed	Here
`oldURL`	Returns the URL of the document before the hash has been changed	Here

JavaScript DOM HashChangeEvent properties:

newURL

[JavaScriptPropertyHashChangeEventNewURL]

The `newURL` property of HashChangeEvent objects within the JavaScript DOM returns the URL of the document after the hash has been changed.

oldURL

[JavaScriptPropertyHashChangeEventOldURL]

The `oldURL` property of HashChangeEvent objects within the JavaScript DOM returns the URL of the document before the hash has been changed.

9. Properties and Methods for PageTransitionEvent objects

[JavaScriptTutorialDOMDetails9]

PageTransitionEvent objects within the JavaScript DOM support the following properties:

Properties (when FocusEvent is triggered):

Property	Description	More
`persisted`	Returns whether the webpage was cached by	Here

	browser	

JavaScript DOM PageTransitionEvent properties:

persisted

[JavaScriptPropertyPageTransitionEventPersisted]

The `persisted` property of PageTransitionEvent objects within the JavaScript DOM returns whether the webpage was cached by browser.

10. Properties and Methods for FocusEvent objects

[Nematrian website page: JavaScriptTutorialDOMDetails10, © Nematrian 2017]

FocusEvent objects within the JavaScript DOM support the following properties:

Properties (when FocusEvent is triggered):

Property	Description	More
`relatedTarget`	Returns element related to the element triggering event	Here

JavaScript DOM FocusEvent properties:

relatedTarget

[JavaScriptPropertyFocusEventRelatedTarget]

The `relatedTarget` property of FocusEvent objects within the JavaScript DOM returns element related to the element triggering the event.

11. Properties and Methods for AnimationEvent objects

[JavaScriptTutorialDOMDetails11]

AnimationEvent objects within the JavaScript DOM support the following properties:

Properties:

Property	Description	More
`animationName`	Returns name of animation	Here
`elapsedTime`	Returns number of seconds animation has been running	Here

JavaScript DOM AnimationEvent properties:

animationName

[JavaScriptPropertyAnimationEventAnimationName]

The `animationName` property of AnimationEvent objects within the JavaScript DOM returns the name of the animation.

elapsedTime

[JavaScriptPropertyAnimationEventElapsedTime]

The `elapsedTime` property of AnimationEvent objects within the JavaScript DOM returns the number of seconds the animation has been running.

12. JavaScript Properties and Methods for TransitionEvent objects

[JavaScriptTutorialDOMDetails12]

TransitionEvent objects within the JavaScript DOM support the following properties:

Properties:

Property	Description	More
`elapsedTime`	Returns number of seconds transition has been running	Here
`propertyName`	Returns name of CSS property associated with transition	Here

JavaScript DOM TransitionEvent properties:

elapsedTime

[JavaScriptPropertyTransitionEventElapsedTime]

The `elapsedTime` property of TransitionEvent objects within the JavaScript DOM returns the number of seconds the transition has been running.

propertyName

[JavaScriptPropertyTransitionEventPropertyName]

The `propertyName` property of TransitionEvent objects within the JavaScript DOM returns the name of CSS property associated with transition.

13. JavaScript Properties and Methods for WheelEvent objects

[JavaScriptTutorialDOMDetails13]

WheelEvent objects within the JavaScript DOM support the following properties:

Properties:

Property	Description	More
`deltaMode`	Returns a number indicating the unit of measurement for delta values	Here
`deltaX`	Returns the horizontal (x-axis) scroll amount of a mouse wheel	Here
`deltaY`	Returns the vertical (y-axis) scroll amount of a mouse wheel	Here
`deltaZ`	Returns the z-axis scroll amount of a mouse wheel	Here

JavaScript DOM WheelEvent properties:

deltaMode

[JavaScriptPropertyWheelEventDeltaMode]

The `deltaMode` property of WheelEvent objects within the JavaScript DOM returns a number indicating the unit of measurement for delta values.

deltaX

[JavaScriptPropertyWheelEventDeltaX]

The `deltaX` property of WheelEvent objects within the JavaScript DOM returns the horizontal (x-axis) scroll amount of a mouse wheel.

deltaY

[JavaScriptPropertyWheelEventDeltaY]

The `deltaY` property of WheelEvent objects within the JavaScript DOM returns the vertical (y-axis) scroll amount of a mouse wheel.

deltaZ

[JavaScriptPropertyWheelEventDeltaZ]

The `deltaZ` property of WheelEvent objects within the JavaScript DOM returns the z-axis scroll amount of a mouse wheel.

14. Properties and Methods for TouchEvent objects

[JavaScriptTutorialDOMDetails14]

TouchEvent objects within the JavaScript DOM support the following properties:

Properties:

Property	Description	More
`altKey`	Returns `true` if 'ALT' key was pressed, otherwise	Here

	returns `false`	
`changedTouches`	Returns a TouchList of all the Touch objects representing those points of contact where state has changed between previous touch event and this one	Here
`ctrlKey`	Returns `true` if 'CTRL' key was pressed, otherwise returns `false`	Here
`metaKey`	Returns `true` if 'META' key was pressed, otherwise returns `false`	Here
`shiftKey`	Returns `true` if 'SHIFT' key was pressed, otherwise returns `false`	Here
`targetTouches`	Returns a TouchList of all the Touch objects that are both currently in contact with touch surface and were also started on same element as the event target	Here
`touches`	Returns a TouchList of all the Touch objects that are currently in contact with touch surface irrespective of target or changed status	Here

JavaScript Properties and Methods for TouchList objects

[JavaScriptTutorialDOMDetailsTouchList]

Some properties of TouchEvent objects within the JavaScript DOM involve TouchLists. These represent a list of contact points with a touch surface. For example, if the user has four fingers touching a screen or trackpad then the corresponding TouchList object would have 4 Touch entries, one for each finger. It supports the following properties and methods:

Properties:

Property	Description	More
`length`	Returns the number of Touch objects in the TouchList	

Methods:

Method	Description	More
`item()`	Returns the Touch object at the specified position in the TouchList	

JavaScript Properties and Methods for Touch objects

[JavaScriptTutorialDOMDetailsTouch]

Some properties of TouchEvent objects within the JavaScript DOM involve Touch objects. These represent a single contact point on a touch-sensitive device. Touch objects support the following properties and methods:

Properties:

Property	Description	More

`clientX`	x-coordinate of touch point relative to left edge of browser viewport	
`clientY`	y-coordinate of touch point relative to left edge of browser viewport	
`identifier`	A unique identified assigned to a touch point, which it will retain for the duration of its movement around the surface	
`pageX`	x-coordinate of touch point relative to left edge of page / document	
`pageY`	y-coordinate of touch point relative to top edge of page / document	
`screenX`	x-coordinate of touch point relative to left edge of screen	
`screenY`	y-coordinate of touch point relative to top edge of screen	
`target`	Element on which the touch point started when first placed on surface	

There are also some (currently) experimental properties such as `radiusX`, `radiusY`, `rotationAngle` (relate to ellipse that most closely describes area of contact between user and surface) and `force` (amount of pressure being applied to surface by the user, between 0 and 1).

JavaScript DOM TouchEvent properties:

altKey

[JavaScriptPropertyTouchEventAltKey]

The `altKey` property of TouchEvent objects within the JavaScript DOM returns `true` if 'ALT' key was pressed, otherwise returns `false`.

changedTouches

[JavaScriptPropertyTouchEventChangedTouches]

The `changedTouches` property of TouchEvent objects within the JavaScript DOM returns a TouchList of all the Touch objects representing those points of contact where the state has changed between previous touch event and this one.

ctrlKey

[JavaScriptPropertyTouchEventCtrlKey]

The `ctrlKey` property of TouchEvent objects within the JavaScript DOM returns `true` if 'CTRL' key was pressed, otherwise returns `false`.

metaKey

[JavaScriptPropertyTouchEventMetaKey]

The `metaKey` property of TouchEvent objects within the JavaScript DOM returns `true` if 'META' key was pressed, otherwise returns `false`.

shiftKey

[JavaScriptPropertyTouchEventShiftKey]

The `shiftKey` property of TouchEvent objects within the JavaScript DOM returns `true` if 'SHIFT' key was pressed, otherwise returns `false`.

targetTouches

[JavaScriptPropertyTouchEventTargetTouches]

The `targetTouches` property of TouchEvent objects within the JavaScript DOM returns a TouchList of all the Touch objects representing those points of contact that are both currently in contact with touch surface and were also started on same element as the event target.

touches

[JavaScriptPropertyTouchEventTouches]

The `touches` property of TouchEvent objects within the JavaScript DOM returns a TouchList of all the Touch objects that are currently in contact with the touch surface irrespective of target or changed status.

I. JavaScript Properties and Methods for Style objects

[JavaScriptTutorialDOMDetailsStyles]

The JavaScript DOM Style object represents an individual style statement. It can be accessed from the head section of a document, using e.g. `document.getElementsByTagName("STYLE")`, or for specific HTML elements, using e.g. `document.getElementsById(`*ElementId*`).style`.

The Style object has properties that largely align with corresponding CSS properties as follows:

CSS property	Style object property	More
`align-content`	`alignContent`	Here
`align-items`	`alignItems`	Here
`align-self`	`alignSelf`	Here
`all`		Here
`animation`	`animation`	Here
`animation-delay`	`animationDelay`	Here
`animation-direction`	`animationDirection`	Here
`animation-duration`	`animationDuration`	Here
`animation-fill-mode`	`animationFillMode`	Here

animation-iteration-count	animationIterationCount	Here
animation-name	animationName	Here
animation-play-state	animationPlayState	Here
animation-timing-function	animationTimingFunction	Here
backface-visibility	backfaceVisibility	Here
background	background	Here
background-attachment	backgroundAttachment	Here
background-blend-mode		Here
background-clip	backgroundClip	Here
background-color	backgroundColor	Here
background-image	backgroundImage	Here
background-origin	backgroundOrigin	Here
background-position	backgroundPosition	Here
background-repeat	backgroundRepeat	Here
background-size	backgroundSize	Here
border	border	Here
border-bottom	borderBottom	Here
border-bottom-color	borderBottomColor	Here
border-bottom-left-radius	borderBottomLeftRadius	Here
border-bottom-right-radius	borderBottomRightRadius	Here
border-bottom-style	borderBottomStyle	Here
border-bottom-width	borderBottomWidth	Here
border-collapse	borderCollapse	Here
border-color	borderColor	Here
border-image	borderImage	Here
border-image-outset	borderImageOutset	Here
border-image-repeat	borderImageRepeat	Here
border-image-	borderImageSlice	Here

slice		
border-image-source	borderImageSourse	Here
border-image-width	borderImageWidth	Here
border-left	borderLeft	Here
border-left-color	borderLeftColor	Here
border-left-style	borderLeftStyle	Here
border-left-width	borderLeftWidth	Here
border-radius	borderRadius	Here
border-right	borderRight	Here
border-right-color	borderRightColor	Here
border-right-style	borderRightStyle	Here
border-right-width	borderRightWidth	Here
border-spacing	borderSpacing	Here
border-style	borderStyle	Here
border-top	borderTop	Here
border-top-color	borderTopColor	Here
border-top-left-radius	borderTopLeftRadius	Here
border-top-right-radius	borderTopRightRadius	Here
border-top-style	borderTopStyle	Here
border-top-width	borderTopWidth	Here
border-width	borderWidth	Here
bottom	bottom	Here
box-shadow	boxShadow	Here
box-sizing	boxSizing	Here
caption-side	captionSide	Here
clear	clear	Here
clip	clip	Here
color	color	Here
column-count	columnCount	Here
column-fill	columnFill	Here
column-gap	columnGap	Here
column-rule	columnRule	Here
column-rule-color	columnRuleColor	Here
column-rule-style	columnRuleStyle	Here
column-rule-width	columnRuleWidth	Here
column-span	columnSpan	Here

column-width	columnWidth	Here
columns	columns	Here
content	content	Here
counter-increment	counterIncrement	Here
counter-reset	counterReset	Here
cursor	cursor	Here
direction	direction	Here
display	display	Here
empty-cells	emptyCells	Here
filter	filter	Here
flex	flex	Here
flex-basis	flexBasis	Here
flex-direction	flexDirection	Here
flex-flow	flexFlow	Here
flex-grow	flexGrow	Here
flex-shrink	flexShrink	Here
flex-wrap	flexWrap	Here
float	cssFloat	Here
font	font	Here
@font-face		Here
font-family	fontFamily	Here
font-size	fontSize	Here
font-size-adjust	fontSizeAdjust	Here
font-stretch	fontStretch	Here
font-style	fontStyle	Here
font-variant	fontVariant	Here
font-weight	fontWeight	Here
hanging-punctuation	hangingPunctuation	Here
height	height	Here
justify-content	justifyContent	Here
@keyframes		Here
left	left	Here
letter-spacing	letterSpacing	Here
line-height	lineHeight	Here
list-style	listStyle	Here
list-style-image	listStyleImage	Here
list-style-position	listStylePosition	Here
list-style-type	listStyleType	Here
margin	margin	Here
margin-bottom	marginBottom	Here
margin-left	marginLeft	Here
margin-right	marginRight	Here

margin-top	marginTop	Here
max-height	maxHeight	Here
max-width	maxWidth	Here
@media		Here
min-height	minHeight	Here
min-width	minWidth	Here
nav-down	navDown	Here
nav-index	navIndex	Here
nav-left	navLeft	Here
nav-right	navRight	Here
nav-up	navUp	Here
opacity	opacity	Here
order	order	Here
orphans	orphans	Here
outline	outline	Here
outline-color	outlineColor	Here
outline-offset	outlineOffset	Here
outline-style	outlineStyle	Here
outline-width	outlineWidth	Here
overflow	overflow	Here
overflow-x	overflowX	Here
overflow-y	overflowY	Here
padding	padding	Here
padding-bottom	paddingBottom	Here
padding-left	paddingLeft	Here
padding-right	paddingRight	Here
padding-top	paddingTop	Here
page-break-after	pageBreakAfter	Here
page-break-before	pageBreakBefore	Here
page-break-inside	pageBreakInside	Here
perspective	perspective	Here
perspective-origin	perspectiveOrigin	Here
position	position	Here
quotes	quotes	Here
resize	resize	Here
right	right	Here
tab-size	tabSize	Here
table-layout	tableLayout	Here
text-align	textAlign	Here
text-align-last	textAlignLast	Here
text-decoration	textDecoration	Here
text-decoration-	textDecorationColor	Here

color		
text-decoration-line	textDecorationLine	Here
text-decoration-style	textDecorationStyle	Here
text-indent	textIndent	Here
text-justify	textJustify	Here
text-overflow	textOverflow	Here
text-shadow	textShadow	Here
text-transform	textTransform	Here
top	top	Here
transform	transform	Here
transform-origin	transformOrigin	Here
transform-style	transformStyle	Here
transition	transition	Here
transition-delay	transitionDelay	Here
transition-duration	transitionDuration	Here
transition-property	transitionProperty	Here
transition-timing-function	transitionTiming-function	Here
unicode-bidi	unicodeBidi	Here
user-select	userSelect	Here
vertical-align	verticalAlign	Here
visibility	visibility	Here
white-space	whiteSpace	Here
widows	widows	Here
width	width	Here
word-break	wordBreak	Here
word-spacing	wordSpacing	Here
word-wrap	wordWrap	Here
z-index	zIndex	Here

II. Creating and Accessing HTML Elements using JavaScript

[HTMLDomElementNames]

More advanced webpages typically use JavaScript to manipulate individual HTML elements on a webpage. For example, HTML <a> (i.e. anchor) elements can be created or accessed using JavaScript as follows:

Create: e.g. `var x = document.createElement("A")`
Access: e.g. `var x = document.getElementById(`*ElementId*`)`

Here the *ElementId* is the id attribute of the element. The `A` is the JavaScript DOM name for an anchor element. Occasionally the most natural way to access an element does not involve its id attribute in which case there are other possible approach, see detail on individual elements.

For some types of elements (e.g. because there will only typically be one of them in any given document, or because they can be accessed via a specific document property) there may be other simpler ways of accessing the element. For example. the following elements might more commonly be accessed as follows:

Element	Alternative ways of accessing them through JavaScript, e.g.
<body>	`var x = document.getElementsByTagName("BODY")[0]` or `var x = document.body`
<head>	`var x = document.getElementsByTagName("HEAD")[0]`
<html>	`var x = document.getElementsByTagName("HTML")[0]` or `var x = document.documentElement`
<iframe>	`var x = window.frames[`*x*`]`
<title>	`var x = document.getElementsByTagName("TITLE")[0]`

Some types of element come in various types, and it is also in practice necessary to set their type when they are created, e.g.:

Element	Steps to create relevant element type
<input>	e.g. `var x = document.createElement("INPUT")` then the type of `<input>` element needs to be set, e.g.: `x.setAttribute("type",` *ElementType*`)` where *ElementType* is e.g. `button` or `checkbox`, …

To add elements that don't reside within any single element inside the document body (such a <datalist> element, you should first create it and then add it to the `document.body` object.

JavaScript DOM object names corresponding to different HTML elements supported by HTML 5 include:

Element	JavaScript DOM name	More
`<a>`	A	Here
`<abbr>`	ABBR	Here
`<address>`	ADDRESS	Here
`<area>`	AREA	Here
`<article>`	ARTICLE	Here
`<aside>`	ASIDE	Here
`<audio>`	AUDIO	Here
`<b>`	B	Here
`<base>`	BASE	Here
`<bdi>`	BDI	Here
`<bdo>`	BDO	Here
`<blockquote>`	BLOCKQUOTE	Here
`<body>`	BODY	Here
` `	BR	Here
`<button>`	BUTTON	Here

<canvas>	CANVAS	Here
<caption>	CAPTION	Here
<cite>	CITE	Here
<code>	CODE	Here
<col>	COL	Here
<colgroup>	COLGROUP	Here
<data>	DATA	Here
<datalist>	DATALIST	Here
<dd>	DD	Here
<del>	DEL	Here
<details>	DETAILS	Here
<dfn>	DFN	Here
<dialog>	DIALOG	Here
	DIV	Here
<dl>	DL	Here
<dt>	DT	Here
	EM	Here
<embed>	EMBED	Here
<fieldset>	FIELDSET	Here
<figcaption>	FIGCAPTION	Here
<figure>	FIGURE	Here
<footer>	FOOTER	Here
<form>	FORM	Here
<h1>	H1	Here
<h2>	H2	Here
<h3>	H3	Here
<h4>	H4	Here
<h5>	H5	Here
<h6>	H6	Here
<head>	HEAD	Here
<header>	HEADER	Here
<hr>	HR	Here
<html>	HTML	Here
<i>	I	Here
<iframe>	IFRAME	Here
<img>	IMG	Here
<input>	INPUT	Here
<ins>	INS	Here
<kbd>	KBD	Here
<keygen>	KEYGEN	Here
<label>	LABEL	Here
<legend>	LEGEND	Here
	LI	Here
<link>	LINK	Here
<main>	MAIN	Here
<map>	MAP	Here
<mark>	MARK	Here
<menu>	MENU	Here
<menuitem>	MENUITEM	Here
<meta>	META	Here

<meter>	METER	Here
<nav>	NAV	Here
<noscript>	NOSCRIPT	Here
<object>	OBJECT	Here
	OL	Here
<optgroup>	OPTGROUP	Here
<option>	OPTION	Here
<output>	OUTPUT	Here
<p>	P	Here
<param>	PARAM	Here
<picture>	PICTURE	Here
<pre>	PRE	Here
<progress>	PROGRESS	Here
<q>	Q	Here
<rp>	RP	Here
<rt>	RT	Here
<ruby>	RUBY	Here
<s>	S	Here
<samp>	SAMP	Here
<script>	SCRIPT	Here
<section>	SECTION	Here
<select>	SELECT	Here
<small>	SMALL	Here
<source>	SOURCE	Here
	SPAN	Here
	STRONG	Here
<style>	STYLE	Here
<sub>	SUB	Here
<summary>	SUMMARY	Here
<sup>	SUP	Here
<table>	TABLE	Here
<tbody>	TBODY	Here
<td>	TD	Here
<textarea>	TEXTAREA	Here
<tfoot>	TFOOT	Here
<th>	TH	Here
<thead>	THEAD	Here
<time>	TIME	Here
<title>	TITLE	Here
<tr>	TR	Here
<track>	TRACK	Here
<u>	U	Here
	UL	Here
<var>	VAR	Here
<video>	VIDEO	Here
<wbr>	WBR	Here

III. Standard DOM properties and methods

[HTMLDomStandardPropertiesMethods]

Applying a property or method to an HTML element involves a command along the lines of e.g.:

element.`click()`

where *element* is the variable corresponding to the HTML element and `click()` is the property or method applied to the element, here a method that simulates a mouse click of the element.

Properties and methods that can be applied to all HTML elements (and to some nodes that are not elements) are set out here:

HTML DOM elements also support all relevant HTML DOM event attributes, properties and methods.

Some DOM properties correspond to HTML attributes that are only applicable to certain types of HTML element. These include:

HTML Attribute	JavaScript DOM property	More
accept	accept	Here
accept-charset	acceptCharset	Here
accesskey	accessKey	Here
action	action	Here
alt	alt	Here
async	async	Here
autocomplete	autocomplete	Here
autofocus	autofocus	Here
autoplay	autoplay	Here
challenge	challenge	Here
charset	charset	Here
checked	checked	Here
cite	cite	Here
class	class	Here
cols	cols	Here
colspan	colspan	Here
content	content	Here
contenteditable	contenteditable	Here
contextmenu	contextmenu	Here
controls	controls	Here
coords	coords	Here
crossorigin	crossorigin	Here
data	data	Here
datetime	datetime	Here
default	default	Here
defer	defer	Here
dir	dir	Here
dirname	dirname	Here
disabled	disabled	Here
download	download	Here
draggable	draggable	Here
dropzone	dropzone	Here

enctype	enctype	Here
for	for	Here
form	form	Here
formaction	formaction	Here
formenctype	formenctype	Here
formmethod	formmethod	Here
formnovalidate	formnovalidate	Here
formtarget	formtarget	Here
headers	headers	Here
height	height	Here
hidden	hidden	Here
high	high	Here
href	href	Here
hreflang	hreflang	Here
http-equiv	httpEquiv	Here
icon	icon	Here
id	id	Here
ismap	ismap	Here
keytype	keytype	Here
kind	kind	Here
label	label	Here
lang	lang	Here
list	list	Here
loop	loop	Here
low	low	Here
manifest	manifest	Here
max	max	Here
maxlength	maxlength	Here
media	media	Here
method	method	Here
min	min	Here
multiple	multiple	Here
muted	muted	Here
name	name	Here
novalidate	novalidate	Here
open	open	Here
optimum	optimum	Here
pattern	pattern	Here
placeholder	placeholder	Here
poster	poster	Here
preload	preload	Here
radiogroup	radiogroup	Here
readonly	readonly	Here
rel	rel	Here
required	required	Here
reversed	reversed	Here
rows	rows	Here
rowspan	rowspan	Here
sandbox	sandbox	Here
scope	scope	Here

scoped	scoped	Here
selected	selected	Here
shape	shape	Here
size	size	Here
sizes	sizes	Here
span	span	Here
spellcheck	spellcheck	Here
src	src	Here
srcdoc	srcdoc	Here
srclang	srclang	Here
srcset	srcset	Here
start	start	Here
step	step	Here
style	style	Here
tabindex	tabindex	Here
target	target	Here
title	title	Here
translate	translate	Here
type	type	Here
usemap	usemap	Here
value	value	Here
width	width	Here
wrap	wrap	Here
xmlns	xmlns	Here

IV. The JavaScript BOM (Browser Object Model)

[JavaScriptBOM]

When a page is opened by a browser it is typically opened:

(1) In a window
(2) On a screen
(3) From a specific URL
(4) By a specific browser
(5) Which may have opened this URL (and others from the same web domain) previously

Objects exposed by the browser within JavaScript can inform the JavaScript programmer about the characteristics of (1) – (5), which can help to provide a more responsive user experience. These objects are collectively known as the JavaScript BOM, i.e. Browser Object Model. There are no agreed standards for these objects, but majr browsers typically implement them:

The window object:

The `window` object represents the open window. If a page contains some <iframe> elements then separate window objects are created by the browser for each <iframe> as well as one for the main page. It typically supports the following properties and methods:

Properties:

Property	Description	More
`closed`	Returns `true` if the window has been closed, `false` otherwise	Here
`defaultStatus`	Sets / returns default text in window statusbar	Here
`document`	Returns the document object currently associated with the window	Here
`frameElement`	Returns the <iframe> object in which the current window resides	Here
`frames`	Returns an array-like object of all <iframe> objects in the current window object	Here
`history`	Returns the history object for the window	Here
`innerHeight`	Returns the height of the window's content area	Here
`innerWidth`	Returns the width of the window's content area	Here
`length`	Returns the number of <iframe> objects in the current window	Here
`localStorage`	Returns a reference to the local storage object used for the object	Here
`location`	Returns the location object for the window	Here
`name`	Sets / returns window name	Here
`navigator`	Returns navigator object for the window	Here
`opener`	Returns the window that created this window	Here
`outerHeight`	Returns the height of the window including toolbars, scrollbars etc.	Here
`outerWidth`	Returns the width of the window including toolbars, scrollbars etc.	Here
`pageXOffset`	Returns number of pixels current document has been scrolled horizontally (from upper left corner of window)	Here
`pageYOffset`	Returns number of pixels current document has been scrolled vertically (from upper left corner of window)	Here
`parent`	Returns parent window of the window	Here
`screen`	Returns screen object for window	Here
`screenLeft`	Returns horizontal coordinate of window relative to screen	Here
`screenTop`	Returns vertical coordinate of window relative to screen	Here
`screenX`	Returns horizontal coordinate of window relative to screen	Here
`screenY`	Returns vertical coordinate of window relative to screen	Here
`sessionStorage`	Stores data in a web browser (one session) in the form of key/value pairs	Here
`scrollX`	Alias for `pageXOffset`	Here
`scrollY`	Alias for `pageYOffset`	Here
`self`	Returns the current window	Here
`status`	Sets / returns text in window statusbar	Here
`top`	Returns topmost browser window	Here

Methods:

Method	Description	More
`alert()`	Displays an alert box	Here
`atob()`	Decodes a base-64 encoded string	Here
`blur()`	Removes focus from window	Here
`btoa()`	Encodes a string in base-64	Here
`clearInterval()`	Clears timer set with `setInterval()`	Here
`clearTimeout()`	Clears timer set with `setTimeout()`	Here
`close()`	Closes current window	Here
`confirm()`	Displays a dialog box (with an OK and Cancel button)	Here
`focus()`	Sets focus to window	Here
`getComputedStyle()`	Gets current computed CSS styles applied to element	Here
`getSelection()`	Returns Selection object representing range of text selected by user	Here
`matchMedia()`	Returns MediaQueryList object representing the results of applying a specified CSS media query string	Here
`moveBy()`	Moves window relative to current position	Here
`moveTo()`	Moves window to a specified position	Here
`open()`	Opens new browser window	Here
`print()`	Prints contents of window	Here
`prompt()`	Displays dialog box prompting user for input	Here
`resizeBy()`	Resizes window by specified numbers of pixels	Here
`resizeTo()`	Resizes windows to specified width and height	Here
`scroll()`	Depreciated (replaced by `scrollTo()` method)	Here
`scrollBy()`	Scrolls document by specified number of pixels	Here
`scrollTo()`	Scrolls document to specified coordinates	Here
`setInterval()`	Calls function or evaluates expression at specified intervals (in milliseconds)	Here
`setTimeout()`	Calls function or evaluates expression after a specified interval (in milliseconds)	Here
`stop()`	Stops window from loading	Here

The screen object:

The `screen` object provides information about the screen in which the browser window has opened. It typically supports the following properties and methods:

Properties:

Property	Description	More
`availHeight`	Returns screen height (excluding taskbar)	Here
`availWidth`	Returns screen width (excluding taskbar)	Here
`colorDepth`	Returns bit depth of colour palette	Here
`height`	Returns total height of screen	Here
`pixelDepth`	Returns colour resolution (bits per pixel) of screen	Here
`width`	Returns total width of screen	Here

The location object:

The `location` object provides information about the URL populating the current window. It typically supports the following properties and methods:

Properties:

Property	Description	More
`hash`	Anchor part of `href` attribute	Here
`host`	Hostname and port part of `href` attribute	Here
`hostname`	Hostname part of `href` attribute	Here
`href`	Sets / returns entire URL	Here
`origin`	Returns protocol, hostname and port part of `href` attribute	Here
`pathname`	Pathname part of `href` attribute	Here
`port`	Port part of `href` attribute	Here
`protocol`	Protocol part of `href` attribute	Here
`search`	Query-string part of `href` attribute	Here
`status`	Sets / Returns text in window statusbar	Here
`top`	Returns topmost browser window	Here

Methods:

Method	Description	More
`assign()`	Loads new document	Here
`reload()`	Reloads document	Here
`replace()`	Replaces current document with new one	Here

The navigator object:

The `navigator` object provides information about the browser that has opened the window. It typically supports the following properties and methods:

Properties:

Property	Description	More
`appCodeName`	Returns browser code name	Here
`appName`	Returns browser name	Here
`appVersion`	Returns browser version information	Here
`cookieEnabled`	Indicates whether cookies are enabled in browser	Here
`geolocation`	Returns Geolocation object (can be used to locate user's position)	Here
`language`	Returns browser language	Here
`online`	Returns whether browser is online	Here
`platform`	Returns platform the browser is compiled for	Here
`product`	Returns browser engine name	Here
`userAgent`	Returns user agent header sent by browser to server	Here

Methods:

Method	Description	More
`javaEnabled()`	Indicates whether browser has Java enabled	Here

The history object:

The `history` object provides information on the URLs visited by the user within the browser. It typically supports the following properties and methods:

Properties:

Property	Description	More
`length`	Returns number of URLs in history list	Here

Methods:

Method	Description	More
`back()`	Loads previous URL in history list	Here
`forward()`	Loads next URL in history list	Here
`go()`	Loads URL in history list specified by index number	Here

Window properties:

closed

[JavaScriptPropertyWindowClosed]

The `closed` property (of the JavaScript BOM window object) returns `true` if the window has been closed or false if it has not been closed. If the window doesn't exist (e.g. because it was never opened) then this can be tested for by e.g. evaluating (!*windowvar*) as this will evaluate to false if *windowvar* does not exist.

defaultStatus

[JavaScriptPropertyWindowDefaultStatus]

The `defaultStatus` property (of the JavaScript BOM window object) sets or returns the default text in the window statusbar.

Setting the `defaultStatus` property typically does not work with many major browsers (as it introduces scope for impersonation of sites). To allow scripts to change the status text, the user must typically alter the configuration settings of the browser.

document

[JavaScriptPropertyWindowDocument]

The `document` property (of the JavaScript BOM window object) returns the document object currently associated with the window.

frameElement

[JavaScriptPropertyWindowFrameElement]

The `frameElement` property (of the JavaScript BOM window object) returns the <iframe> object in which the current window resides. If the window is not within an <iframe> object then this property will return `null`.

frames

[JavaScriptPropertyWindowFrames]

The `frames` property (of the JavaScript BOM window object) returns an array-like object of all <iframe> objects in the current window object. The first element has an index entry of 0. The number of <iframe> objects contained in the object can be identified from `frames.length`.

history

[JavaScriptPropertyWindowHistory]

The `history` property (of the JavaScript BOM window object) returns the history object for the window.

innerHeight

[JavaScriptPropertyWindowInnerHeight]

The `innerHeight` property (of the JavaScript BOM window object) returns the height of the window's content area.

innerWidth

[JavaScriptPropertyWindowInnerWidth]

The `innerWidth` property (of the JavaScript BOM window object) returns the width of the window's content area.

length

[JavaScriptPropertyWindowLength]

The `length` property (of the JavaScript BOM window object) returns the number of <iframe> objects in the current window.

localStorage

[JavaScriptPropertyWindowLocalStorage]

The `localStorage` property (of the JavaScript BOM window object) returns a reference to the local storage object in which it is possible to store data within a web browser (permanently) in the form of key/value pairs.

location

[JavaScriptPropertyWindowLocation]

The `location` property (of the JavaScript BOM window object) returns the location object for the window.

name

[JavaScriptPropertyWindowName]

The `name` property (of the JavaScript BOM window object) sets / returns the window name.

navigator

[JavaScriptPropertyWindowNavigator]

The `navigator` property (of the JavaScript BOM window object) returns the navigator object for the window.

opener

[JavaScriptPropertyWindowOpener]

The `opener` property (of the JavaScript BOM window object) returns the window that created this window.

outerHeight

[JavaScriptPropertyWindowOuterHeight]

The `outerHeight` property (of the JavaScript BOM window object) returns the height of the window including toolbars, scrollbars etc.

outerWidth

[JavaScriptPropertyWindowOuterWidth]

The `outerWidth` property (of the JavaScript BOM window object) returns the width of the window including toolbars, scrollbars etc.

pageXOffset

[JavaScriptPropertyWindowPageXOffset]

The `pageXOffset` property (of the JavaScript BOM window object) returns the number of pixels current document has been scrolled horizontally (from upper left corner of window).

pageYOffset

[JavaScriptPropertyWindowPageYOffset]

The `pageYOffset` property (of the JavaScript BOM window object) returns the number of pixels current document has been scrolled vertically (from upper left corner of window).

parent

[JavaScriptPropertyWindowParent]

The `parent` property (of the JavaScript BOM window object) returns the parent window of the window.

screen

[JavaScriptPropertyWindowScreen]

The `screen` property (of the JavaScript BOM window object) returns the screen object for window.

screenLeft

[JavaScriptPropertyWindowScreenLeft]

The `screenLeft` property (of the JavaScript BOM window object) returns the horizontal coordinate of window relative to screen.

screenTop

[JavaScriptPropertyWindowScreenTop]

The `screenTop` property (of the JavaScript BOM window object) returns the vertical coordinate of window relative to screen.

screenX

[JavaScriptPropertyWindowScreenX]

The `screenX` property (of the JavaScript BOM window object) returns the horizontal coordinate of window relative to screen.

screenY

[JavaScriptPropertyWindowScreenY]

The `screenY` property (of the JavaScript BOM window object) returns the vertical coordinate of window relative to screen.

scrollX

[JavaScriptPropertyWindowScrollX]

The `scrollX` property (of the JavaScript BOM window object) is an alias for the for pageXOffset property.

scrollY

[JavaScriptPropertyWindowScrollY]

The `scrollY` property (of the JavaScript BOM window object) is an alias for the for pageYOffset property.

self

[JavaScriptPropertyWindowSelf]

The `self` property (of the JavaScript BOM window object) returns the current window.

sessionStorage

[JavaScriptPropertyWindowSessionStorage]

The `sessionStorage` property (of the JavaScript BOM window object) returns a reference to the session storage object in which it is possible to store data within a web browser (temporarily, for a single session) in the form of key/value pairs.

status

[JavaScriptPropertyWindowStatus]

The `status` property (of the JavaScript BOM window object) sets / returns the text in the window status bar.

Setting the `status` property typically does not work with many major browsers (as it introduces scope for impersonation of sites). To allow scripts to change the status text, the user must typically alter the configuration settings of the browser.

top

[JavaScriptPropertyWindowTop]

The `top` property (of the JavaScript BOM window object) returns the topmost browser window.

Window methods:

alert()

[website page: JavaScriptMethodWindowAlert]

The `alert()` method (when applied to Window objects in the JavaScript BOM) displays text in an alert box.

It has the following syntax with the following parameters. It does not return a value.

`window.alert(`*message*`)`

Parameter	Required / Optional	Description
message	Optional	String specifying text to display in an alert box

atob()

[JavaScriptMethodWindowAtob]

The `atob()` method (when applied to Window objects in the JavaScript BOM) decodes a base-64 encoded string (encoded using the btoa() method.

It has the following syntax with the following parameters. It returns the decoded string.

`window.atob(`*str*`)`

Parameter	Required / Optional	Description
str	Required	String which has been encoded using the btoa() method

blur()

[JavaScriptMethodWindowBlur]

The `blur()` method (when applied to Window objects in the JavaScript BOM) removes focus from the window.

It has the following syntax with no parameters. It does not return a value.

`window.blur(`*message*`)`

btoa()

[JavaScriptMethodWindowBtoa]

The `btoa()` method (when applied to Window objects in the JavaScript BOM) encodes a string into base-64, using A-Z, a-z, 0-9, “+”, “/” and “=” characters to encode the string. The string can be decoded using the atob() method.

It has the following syntax with the following parameters. It returns the encoded string.

`window.btoa(`*str*`)`

Parameter	Required / Optional	Description
str	Required	String to be base-64 encoded

clearInterval()

[JavaScriptMethodWindowClearInterval]

The `clearInterval()` method (when applied to Window objects in the JavaScript BOM) clears a timer set using the setInterval() method.

It has the following syntax with the following parameters. It does not return a value.

```
window.clearInterval(id)
```

Parameter	Required / Optional	Description
id	Required	The id of the timer returned by the setInterval() method

clearTimeout()

[JavaScriptMethodWindowClearTimeout]

The `clearTimeout()` method (when applied to Window objects in the JavaScript BOM) clears a timer set using the setTimeout() method.

It has the following syntax with the following parameters. It does not return a value.

```
window.clearTimeout(id)
```

Parameter	Required / Optional	Description
id	Required	The id of the timer returned by the setTimeout() method

close()

[JavaScriptMethodWindowClose]

The `close()` method (when applied to Window objects in the JavaScript BOM) closes the current window.

It has the following syntax with no parameters. It does not return a value.

```
window.close()
```

confirm()

[JavaScriptMethodWindowConfirm]

The `confirm()` method (when applied to Window objects in the JavaScript BOM) displays text in a dialog box (with an OK and Cancel button).

It has the following syntax with the following parameters. It returns true if the user clicks the OK button, otherwise false.

`window.confirm(`*message*`)`

Parameter	Required / Optional	Description
message	Optional	String specifying text to display in dialog box

focus()

[JavaScriptMethodWindowFocus]

The `focus()` method (when applied to Window objects in the JavaScript BOM) sets focus to the window, which typically brings the window to the foreground (although this may not work as expected in all browsers depending on what settings the user has adopted).

It has the following syntax with no parameters. It does not return a value.

`window.focus(`*message*`)`

getComputedStyle()

[JavaScriptMethodWindowGetComputedStyle]

The `getComputedStyle()` method (when applied to Window objects in the JavaScript BOM) returns the current computed CSS styles applied to a specified element.

It has the following syntax with the following parameters. It returns a CSSStyleDeclaration object.

`window.getComputedStyle(`*element, pseudoelement*`)`

Parameter	Required / Optional	Description
element	Required	Element to get computed style for
pseudoelement	Optional	Pseudo-element

getSelection()

[JavaScriptMethodWindowGetSelection]

The `getSelection()` method (when applied to Window objects in the JavaScript BOM) returns an object representing the range of text selected by user.

Note: one way of returning the text selected is to cast the result to a string (either by appending an empty string or by applying the `toString()` method to the object).

It has the following syntax with no parameters. It returns a Selection object.

`window.getSelection()`

matchMedia()

[JavaScriptMethodWindowMatchMedia]

The `matchMedia()` method (when applied to Window objects in the JavaScript BOM) returns a MediaQueryList object representing the results of applying a specified CSS media query string.

It has the following syntax with the following parameters. It returns a MediaQueryList object.

`window.matchMedia(`*mediaquerystring*`)`

Parameter	Required / Optional	Description
mediaquerystring	Required	String representing media query. This can be any media features that can be included in a CSS @media rule

A MediaQueryList object has two properties and two methods:

Properties:

Property	Description	More
`matches`	Returns `true` if document matches the specified media query list, `false` otherwise	
`media`	Returns a string representing the serialised media query list	

Methods:

Method	Description	More
`addEventListener()`	Adds new listener function, evaluated whenever media query's evaluated result changes	
`removeListener()`	Removes previously added listener function (or does nothing if listener function was not present	

moveBy()

[JavaScriptMethodWindowMoveBy]

The `moveBy()` method (when applied to Window objects in the JavaScript BOM) moves a window by specified amounts (in the x and y directions) relative to its current position.

It has the following syntax with the following parameters. It does not return a value.

`window.moveBy(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	Positive or negative number specifying number of pixels to move window horizontally
y	Required	Positive or negative number specifying number of pixels to move window vertically

moveTo()

[JavaScriptMethodWindowMoveTo]

The `moveTo()` method (when applied to Window objects in the JavaScript BOM) moves a window to a position specified by the x and y coordinates of its left top corner.

It has the following syntax with the following parameters. It does not return a value.

`window.moveTo(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	Positive or negative number specifying number of pixels horizontally
y	Required	Positive or negative number specifying number of pixels vertically

open()

[JavaScriptMethodWindowOpen]

The `open()` method (when applied to Window objects in the JavaScript BOM) opens a new browser window.

It has the following syntax with the following parameters. It does not return a value.

`window.open(`*URL, name, specifications, replace*`)`

Parameter	Required / Optional	Description
URL	Optional	URL of page to open. If no URL is specified then a new window with `about:blank` is opened
name	Optional	The HTML target attribute (name) applicable to the window (e.g. `_blank`, …)
specifications	Optional	A comma-separated list of items, no whitespaces, see below
replace	Optional	Boolean specifying whether URL replaces the current entry in the history list (`true`) or creates a new entry (`false`)

Values supported by the *specifications* parameter vary by browser but for some browsers include following:

Sub-Parameter	Options	Description
`channelmode`	`yes|no|1|0`	Whether to display window in 'theatre' mode (default no)
`directories`	`yes|no|1|0`	Obsolete. Whether to add directory buttons (default yes)
`fullscreen`	`yes|no|1|0`	Whether to display in full-screen mode (defatul no)
`height`	*pixels*	Height of window (min 100)
`left`	*pixels*	Left position of window (min 0)

`location`	`yes|no|1|0`	Whether to display address field
`menubar`	`yes|no|1|0`	Whether to display menubar
`resizable`	`yes|no|1|0`	Whether window is resizable
`scrollbars`	`yes|no|1|0`	Whether to display scrollbars
`titlebar`	`yes|no|1|0`	Whether to display titlebar. Ignored unless calling application is HTML Application or trusted dialog box
`toolbar`	`yes|no|1|0`	Whether to display browser toolbar
`top`	*pixels*	top position of window (min 0)
`width`	*pixels*	Width of window (min 100)

print()

[JavaScriptMethodWindowPrint]

The `print()` method (when applied to Window objects in the JavaScript BOM) prints the contents of the window.

It has the following syntax with no parameters. It does not return a value.

```
window.print()
```

prompt()

[JavaScriptMethodWindowPrompt]

The `prompt()` method (when applied to Window objects in the JavaScript BOM) displays text in a dialog box prompting user for input (and with an OK and Cancel button).

It has the following syntax with the following parameters. It returns a string if the user clicks OK, being the input value (an empty string if the user didn't input anything), or `null` if the user clicks cancel.

`window.prompt(`*text, defaulttext*`)`

Parameter	Required / Optional	Description
text	Required	String specifying text to display in dialog box
defaulttext	Optional	String specifying default input text

resizeBy()

[JavaScriptMethodWindowResizeBy]

The `resizeBy()` method (when applied to Window objects in the JavaScript BOM) resizes a window by specified amounts (in the x and y directions) leaving the position of the top left corner unchanged.

It has the following syntax with the following parameters. It does not return a value.

`window.resizeBy(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	Positive or negative number specifying number of pixels to change width by
y	Required	Positive or negative number specifying number of pixels to change height by

resizeTo()

[JavaScriptMethodWindowResizeTo]

The `resizeTo()` method (when applied to Window objects in the JavaScript BOM) resizes a window to a specified size (in the x and y directions) leaving the position of the top left corner unchanged.

It has the following syntax with the following parameters. It does not return a value.

```
window.resizeTo(x, y)
```

Parameter	Required / Optional	Description
x	Required	Number specifying width in pixels
y	Required	Number specifying height in pixels

scroll()

[JavaScriptMethodWindowScroll]

The `scroll()` method (when applied to Window objects in the JavaScript BOM) scrolls the document to specified coordinates. It is depreciated (replaced by the scrollTo() method)

It has the following syntax with the following parameters. It does not return a value.

```
window.scroll(x, y)
```

Parameter	Required / Optional	Description
x	Required	Number of pixels to scroll to, along horizontal axis
y	Required	Number of pixels to scroll to, along vertical axis

scrollBy()

[JavaScriptMethodWindowScrollBy]

The `scrollBy()` method (when applied to Window objects in the JavaScript BOM) scrolls the document by specified number of pixels.

Note: the `visible` property of the window's scrollbar needs to be set to true for this method to work.

It has the following syntax with the following parameters. It does not return a value.

```
window.scrollBy(x, y)
```

Parameter	Required / Optional	Description
x	Required	Positive or negative number specifying number of pixels to scroll by (positive causes scroll to right, negative scroll to the left)
y	Required	Positive or negative number specifying number of pixels to scroll by (positive causes scroll down, negative scroll up)

scrollTo()

[JavaScriptMethodWindowScrollTo]

The `scrollTo()` method (when applied to Window objects in the JavaScript BOM) scrolls the document to specified coordinates.

It has the following syntax with the following parameters. It does not return a value.

```
window.scrollTo(x, y)
```

Parameter	Required / Optional	Description
x	Required	Number of pixels to scroll to, along horizontal axis
y	Required	Number of pixels to scroll to, along vertical axis

setInterval()

[JavaScriptMethodWindowSetInterval]

The `setInterval()` method (when applied to Window objects in the JavaScript BOM) calls a function or evaluates an expression at specified intervals (in milliseconds). It will continue calling the function until the clearInterval() method is called or until the window is closed.

It has the following syntax with the following parameters. It returns an id value (number) which is then used as the parameter for the clearInterval() method.

Note: use the setTimeout() method to execute the function only once.

```
window.setInterval(function, milliseconds, param1, param2, …)
```

Parameter	Required / Optional	Description
function	Required	Function to be evaluated
milliseconds	Required	Interval (in milliseconds) between consecutive executions (if less than 10 then defaulted to 10)
param1, param2, …	Optional	Additional parameters passed to function

setTimeout()

[JavaScriptMethodWindowSetTimeout]

The `setTimeout()` method (when applied to Window objects in the JavaScript BOM) calls a function or evaluates expression (once) after a specified interval (in milliseconds).

It has the following syntax with the following parameters. It returns an id value (number) which is then used as the parameter for the clearTimeout() method.

Note: use the setInterval () method to execute the function repeatedly.

`window.setTimeout(`*function, milliseconds, param1, param2, …*`)`

Parameter	Required / Optional	Description
function	Required	Function to be evaluated
milliseconds	Required	Interval (in milliseconds) between consecutive executions (if less than 10 then defaulted to 10)
param1, param2, …	Optional	Additional parameters passed to function

stop()

[JavaScriptMethodWindowStop]

The `stop()` method (when applied to Window objects in the JavaScript BOM) stops the window from loading.

It has the following syntax with no parameters. It does not return a value.

`window.stop(`*message*`)`

Screen properties:

availHeight

[JavaScriptPropertyScreenAvailHeight]

The `availHeight` property (of the JavaScript BOM screen object) returns the screen height (excluding the taskbar).

availWidth

[JavaScriptPropertyScreenAvailWidth]

The `availWidth` property (of the JavaScript BOM screen object) returns the screen width (excluding the taskbar).

colorDepth

[JavaScriptPropertyScreenColorDepth]

The `colorDepth` property (of the JavaScript BOM screen object) returns the bit depth of the colour palette.

height

[JavaScriptPropertyScreenHeight]

The `height` property (of the JavaScript BOM screen object) returns the total height of the screen.

pixelDepth

[JavaScriptPropertyScreenPixelDepth]

The `pixelDepth` property (of the JavaScript BOM screen object) returns the colour resolution (bits per pixel) of the screen.

width

[JavaScriptPropertyScreenWidth]

The `width` property (of the JavaScript BOM screen object) returns the total width of the screen.

Screen Methods:

N/A

Location properties:

hash

[JavaScriptPropertyLocationHash]

The `hash` property (of the JavaScript BOM location object) returns the anchor part of the `href` attribute.

host

[JavaScriptPropertyLocationHost]

The `host` property (of the JavaScript BOM location object) returns the hostname and port part of the `href` attribute.

hostname

[JavaScriptPropertyLocationHostname]

The `hostname` property (of the JavaScript BOM location object) returns the hostname part of the `href` attribute.

href

[JavaScriptPropertyLocationHref]

The `href` property (of the JavaScript BOM location object) sets / returns the entire URL.

origin

[JavaScriptPropertyLocationOrigin]

The `origin` property (of the JavaScript BOM location object) returns the protocol, hostname and port part of the `href` attribute.

pathname

[JavaScriptPropertyLocationPathname]

The `pathname` property (of the JavaScript BOM location object) returns the pathname part of the `href` attribute.

port

[JavaScriptPropertyLocationPort]

The `port` property (of the JavaScript BOM location object) returns the port part of the `href` attribute.

protocol

[JavaScriptPropertyLocationProtocol]

The `protocol` property (of the JavaScript BOM location object) returns the protocol part of the `href` attribute.

search

[JavaScriptPropertyLocationSearch]

The `search` property (of the JavaScript BOM location object) returns the query-string part of the `href` attribute.

status

[JavaScriptPropertyLocationStatus]

The `status` property (of the JavaScript BOM location object) sets / returns the text in the window statusbar.

top

[JavaScriptPropertyLocationTop]

The `top` property (of the JavaScript BOM location object) returns the topmost browser window.

Location methods:

assign()

[JavaScriptMethodLocationAssign]

The `assign()` method (when applied to Location objects in the JavaScript BOM) loads new document (but in a way that still allows the back button of the browser to go back to the original document).

It has the following syntax with the following parameters. It does not return a value.

location`.assign(`*URL*`)`

Parameter	Required / Optional	Description
URL	Required	URL of page to navigate to

reload()

[JavaScriptMethodLocationReload]

The `reload()` method (when applied to Location objects in the JavaScript BOM) reloads the current document.

It generally does the same as the browser's reload button. However, it is possible to specify where the reload comes from, see below.

It has the following syntax with the following parameters. It does not return a value.

location`.reload(`*get*`)`

Parameter	Required / Optional	Description
get	Optional	Boolean specifying whether to reload the current page from the server (`true`) or from the cache (`false`, is the default)

replace()

[JavaScriptMethodLocationReplace]

The `replace()` method (when applied to Location objects in the JavaScript BOM) replaces the current document with a new document (in a way that removes the URL of the current document from the document history, so stopping the back button of the browser going back to the original document).

It has the following syntax with the following parameters. It does not return a value.

location`.replace(`*URL*`)`

Parameter	Required / Optional	Description
URL	Required	URL of page to navigate to

Navigator properties:

appCodeName

[JavaScriptPropertyNavigatorAppCodeName]

The `appCodeName` property (of the JavaScript BOM navigator object) returns the browser code name.

appName

[JavaScriptPropertyNavigatorAppName]

The `appName` property (of the JavaScript BOM navigator object) returns the browser name.

appVersion

[JavaScriptPropertyNavigatorAppVersion]

The `appVersion` property (of the JavaScript BOM navigator object) returns browser version information.

cookieEnabled

[JavaScriptPropertyNavigatorCookieEnabled]

The `cookieEnabled` property (of the JavaScript BOM navigator object) indicates whether cookies are enabled in the browser.

geolocation

[JavaScriptPropertyNavigatorGeolocation]

The `geolocation` property (of the JavaScript BOM navigator object) returns a Geolocation object (which can be used to locate the user's position).

language

[JavaScriptPropertyNavigatorLanguage]

The `language` property (of the JavaScript BOM navigator object) returns the browser language.

online

[JavaScriptPropertyNavigatorOnline]

The `online` property (of the JavaScript BOM navigator object) returns whether the browser is online.

platform

[JavaScriptPropertyNavigatorPlatform]

The `platform` property (of the JavaScript BOM navigator object) returns the platform the browser is compiled for.

product

[JavaScriptPropertyNavigatorProduct]

The `product` property (of the JavaScript BOM navigator object) returns the browser engine name.

userAgent

[JavaScriptPropertyNavigatorUserAgent]

The `userAgent` property (of the JavaScript BOM navigator object) returns the user agent header sent by the browser to the server.

Navigator methods:

javaEnabled()

[JavaScriptMethodNavigatorJavaEnabled]

The `javaEnabled()` method (when applied to Navigator objects in the JavaScript BOM) indicates whether the browser has Java enabled.

It has the following syntax with no parameters. It returns `true` if the browser has Java enabled, otherwise `false`.

```
navigator.javaEnabled()
```

History properties:

length

[JavaScriptPropertyHistoryLength]

The `length` property (of the JavaScript BOM history object) returns the number of URLs in the history list.

History methods:

back()

[JavaScriptMethodHistoryBack]

The `back()` method (when applied to History objects in the JavaScript BOM) loads the previous URL in the history list. It is the same as clicking the back button in the browser or applying the `history.go(-1)` method.

It has the following syntax with no parameters. It does not return a value.

```
history.back()
```

forward()

[JavaScriptMethodHistoryForward]

The `forward()` method (when applied to History objects in the JavaScript BOM) loads the next URL in the history list. It is the same as clicking the forward button in the browser or applying the `history.go(1)` method.

It has the following syntax with no parameters. It does not return a value.

```
history.forward()
```

go()

[JavaScriptMethodHistoryGo]

The `go()` method (when applied to History objects in the JavaScript BOM) loads a URL from the history list.

It has the following syntax with the following parameters. It does not return a value.

`history.go(`*param*`)`

Parameter	Required / Optional	Description
param	Required	Either a number indicating where within the history list to go to (e.g. -1 goes back a page, +1 goes forward a page), or a string representing a partial or full URL (method will go to first URL that matches the string)

V. The JavaScript XML DOM

[JavaScriptTutorialXMLDOM]

The JavaScript DOM data structure has a form akin to an XML document, with a tree-like structure that includes nodes at different branching points. This means that some additional methods and properties are available to certain of its components.

Different NodeTypes in an XML DOM

NodeType	Provides / represents	Children	nodeName returns	nodeValue returns
Attr	An attribute	Text,EntityReference	Attribute name	Attribute value
CDATASection	A CDATA section in a document	N/A	#cdata-section	Content of node
Comment	A comment	N/A	#comment	Comment text
Document	Entire document	Element (at most 1), ProcessingInstruction, Comment, DocumentType	#document	null
DocumentFragment	A portion of a document	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference	#document fragment	null
DocumentType	An interface to entities defined for document	None	Doctype name	null
Element	An element	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference	Element name	null
Entity	Entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference	Entity name	null
EntityReference			Entity reference name	null
Notation	Notation declared in the DTD		Notation name	null
ProcessingInstruction			target	Content of node
Text	Textual content in an element or attribute	N/A	#text	Content of node

Each NodeType has a specific name constant and number used for some purposes within the DOM:

NodeType	As number	Named Constant
Attr	2	ATTRIBUTE_NODE
CDATASection	4	CDATA_SECTION_NODE
Comment	8	COMMENT_NODE
Document	9	DOCUMENT_NODE

DocumentFragment	11	DOCUMENT_FRAGMENT_NODE
DocumentType	10	DOCUMENT_TYPE_NODE
Element	1	ELEMENT_NODE
Entity	6	ENTITY_NODE
EntityReference	5	ENTITY_REFERENCE_NODE
Notation	12	NOTATION_NODE
ProcessingInstruction	7	PROCESSING_INSTRUCTION_NODE
Text	3	TEXT_NODE

Node objects

Nodes have the following properties and methods. In the two tables below, we generally illustrate these properties by reference either to the `document` object or by reference to the DOM object equivalents of HTML elements.

Property	Description	More
`attributes`	Returns NamedNodeMap of attributes	Here
`baseURI`	Returns absolute base URI	Here
`childNodes`	Returns collection of the child elements (including text and comment nodes)	Here
`firstChild`	Returns first child node	Here
`lastChild`	Returns last child node	Here
`nextSibling`	Returns next node at same node tree level	Here
`nodeName`	Returns node name	Here
`nodeType`	Returns node type	Here
`nodeValue`	Sets/returns node value	Here
`ownerDocument`	Returns root element (i.e. document object) within which element resides	Here
`parentNode`	Returns parent node	Here
`prefix`	Sets / returns the namespace prefix of a node	
`previousSibling`	Returns previous node at same node tree level	Here
`textContent`	Sets / returns the text content of a node and its descendants	Here

Method	Description	More
`appendChild()`	Adds a new child after last existing one	Here
`cloneNode()`	Clones an element	Here
`compareDocumentPosition()`	Compares position in document of two elements	Here
`contains()`	Returns `true` if node is a descendant of other node, otherwise returns `false`	Here
`focus()`	Gives focus to element	Here
`getFeature`		
`getUserData`		
`hasAttributes()`	Returns `true` if element has any attributes, otherwise returns `false`	Here
`hasChildNodes()`	Returns `true` if element has any child nodes, otherwise returns `false`	Here
`insertBefore()`	Inserts new child node before specific existing child node	Here

`isDefaultNamespace()`	Returns `true` if a specified namespace URI is the default namespace, otherwise returns `false`	Here
`isEqualNode()`	Returns `true` if two elements / nodes are 'equal' (but not necessarily exactly the same), otherwise returns `false`	Here
`isSameNode()`	Returns `true` if two elements / nodes are the same (i.e. equal but also computationally refer to the same node), otherwise returns `false`	Here
`lookupNamespaceURI()`		
`normalize()`	Removes empty text nodes and joins adjacent notes	Here
`removeChild()`	Removes specified child node	Here
`replaceChild()`	Replaces specified child node	Here
`setUserData`		

NodeList objects

The NodeList object is the collection of child nodes within a node. It has the following properties and methods:

Property	Description	More
`length`	Returns number of nodes in a NodeList	Here

Method	Description	More
`item()`	Returns node at specified index position in a NodeList	Here

NamedNodeMap objects

These are covered here.

XML Document objects

Many of the properties applicable to XML DOM objects are applicable to the DOM document object more generally and are given here (or here, in relation to the document being itself a DOM element). XML specific properties and methods are set out below.

Property	Description	More
`documentURI`	Sets / returns URI of document	
`xmlEncoding`	Returns XML encoding of document	
`xmlStandalone`	Sets / returns whether document standalone	
`xmlVersion`	Sets / returns XML version of document	

Method	Description	More
`createCDATASection()`	Creates a CDATA section node	
`createEntityReference()`	Clones an element	
`createProcessingInstruction()`	Compares position in document of two elements	

DocumentType objects

Each Document has a DOCTYPE attribute, which is either null or a DocumentType object. It has the following properties and methods:

Property	Description	More
`name`	Returns DTD's name	
`publicId`	Returns DTD's public identifier	
`systemId`	Returns DTD's system identifier	

DOMImplementation objects

The DOMImplementation object performs operations that are independent of any specific instance of the DOM, i.e. it generally tells the code something about the system surrounding the specific document. It has the following properties and methods:

Method	Description	More
`createDocument()`	Creates a new DOM Document object of the specified doctype	
`createDocumentType()`	Creates an empty DocumentType node	
`getFeature()`	Returns an object that implements the API's of the specified feature and version, if there is such an API	
`hasFeature()`	Checks whether the DOM implementation implements a specific feature	

The `hasFeature()` method ought in principle to be very useful for browser feature detection. However, apparently it was inconsistently implemented in different browsers, so its use is no longer recommended.

ProcessingInstruction objects

The ProcessingInstruction object represents a processing instruction. This is a way of keeping processor-specific information in the text of the XML document. It has the following properties:

Property	Description	More
`data`	Sets / returns content of ProcessingInstruction	
`publicId`	Returns target of processing instruction	

DOM element objects

DOM elements have properties and methods shown here.

DOM attribute objects

The Attr object representing DOM attributes has properties and methods shown here. However, it is worth noting that an XML attribute does not have a parent node and is not considered to be a child node of an element. As a result, it returns null for many Node properties.

DOM text objects

Text objects represent textual nodes. They have the following properties and methods:

Property	Description	More
`data`	Sets / returns text of element or attribute	
`isElementContentWhitespace`	Returns true if content is whitespace, otherwise false	
`length`	Length of text of element or attribute	
`wholeText`	Returns all text of text nodes adjacent to this node, concatenated together	

Method	Description	More
`appendData()`	Appends data to node	
`deleteData()`	Deletes data from node	
`insertData()`	Inserts data into node	
`replaceData()`	Replaces data in node	
`replaceWholeText()`	Replaces text in this node and adjacent text nodes with specified text	
`splitText`	Splits node into two at specified offset, and returns new node containing text after offset	
`substringData()`	Extracts data from node	

CDATASection objects

The CDATASection object represents a CDATA section in a document. This contains text that will not be parsed by the parser (i.e. is not treated as markup. The only delimiter recognised in such a section is "]]>" which indicates the end of the section. CDATA sections cannot be nested. They have the following properties and methods:

Property	Description	More
`data`	Sets / returns text of node	
`length`	Length of text of node	

Method	Description	More
`appendData()`	Appends data to node	
`deleteData()`	Deletes data from node	
`insertData()`	Inserts data into node	
`replaceData()`	Replaces data in node	
`splitText`	Splits node into two at specified offset, and returns new node containing text after offset	
`substringData()`	Extracts data from node	

Comment objects

The Comment object represents the content of comment nodes. It has the following properties and methods:

Property	Description	More
`data`	Sets / returns text of node	
`length`	Length of text of node	

Method	Description	More

`appendData()`	Appends data to node	
`deleteData()`	Deletes data from node	
`insertData()`	Inserts data into node	
`replaceData()`	Replaces data in node	
`splitText`	Splits node into two at specified offset, and returns new node containing text after offset	
`substringData()`	Extracts data from node	

XMLHttpRequest objects

The XMLHttpRequest object allows JavaScript to update parts of a web page without reloading the whole page. It also allows the developer to arrange:

- Request data from a server after the page has loaded
- Receive data from a server after the page has loaded
- Send data to a server in the background

It has the following properties and methods:

Property	Description	More
`onreadystatechange`	Function to be called automatically each time the readyState property changes	
`readyState`	Holds status of XMLHttpRequest. Changes from 0 to 4: 0: request not initialised 1: server connection established 2: request received 3: processing request 4: request finished and response ready	
`responseText`	Returns response data as string	
`responseXML`	Returns response data as XML data	
`status`	Returns status number (e.g. “200” for OK, “404” for “Not found”	
`statusText`	Returns status text (e.g. “OK” or “Not found”	

Method	Description	More
`appendData()`	Appends data to node	
`deleteData()`	Deletes data from node	
`insertData()`	Inserts data into node	
`replaceData()`	Replaces data in node	
`splitText`	Splits node into two at specified offset, and returns new node containing text after offset	
`substringData()`	Extracts data from node	

Appendix Z: Further JavaScript Properties and Methods

Global properties:

Infinity

[JavaScriptPropertyGlobalInfinity]

The JavaScript Global `Infinity` property returns `Infinity` (i.e. larger than the upper limit of floating point numbers in the JavaScript language. Note: `-Infinity` arises if the number is negative and exceeds the lower limit of floating point numbers.

It has the following syntax:

```
Infinity
```

NaN

[JavaScriptPropertyGlobalNaN]

The JavaScript Global `NaN` property returns `NaN` (i.e. 'not a number').

It has the following syntax:

```
NaN
```

undefined

[JavaScriptPropertyGlobalUndefined]

The JavaScript Global `undefined` property indicates that a variable has been created but has not yet been defined a valu.

It has the following syntax:

```
undefined
```

Global methods / functions:

Boolean()

[JavaScriptMethodGlobalBoolean]

The JavaScript Global `Boolean()` method converts an object to a Boolean representing the value of the object. If the parameter value is omitted or is 0, -0, false, NaN, undefined, an empty string or the `document.all` DOM object then the method evaluates to `false`. All other parameter values (including the string "false"!) evaluate to `true`.

It has the following syntax with the following parameters:

```
Boolean(x)
```

Parameter	Required / Optional	Description
x	Optional	Input parameter. If missing, then returns `false`.

Note: it is easy to confuse the primitive Boolean values true and false with the values of the Boolean object. For example, any object whose value is not undefined or null evaluates to true when passed to a conditional statement. So, the following statements will result in the *code* being evaluated:

```
var x = new Boolean(false);
if (x) { code }
```

whereas it will not be executed with the following statements:

```
var x = false;
if (x) { code }
```

The output of the global `Boolean` method can also be quite confusing at it involves a type coercion that does not always behave intuitively.

decodeURI()

[JavaScriptMethodGlobalDecodeURI]

The JavaScript Global `decodeURI()` method inverts the outcome of encoding a string using encodeURI. It is depreciated, and decodeURI or decodeURIComponent should be used instead.

It has the following syntax with the following parameters:

```
decodeURI(encodedURI)
```

Parameter	Required / Optional	Description
encodedURI	Required	String representing an encoded URI

More detail on URI encoding is given here.

decodeURIComponent()

[JavaScriptMethodGlobalDecodeURIComponent]

The JavaScript Global `decodeURIcomponent()` method inverts the outcome of encoding a string using encodeURIComponent.

It has the following syntax with the following parameters:

```
decodeURIComponent(encodedURI)
```

Parameter	Required / Optional	Description
encodedURI	Required	String representing an encoded URI

More detail on URI encoding is given here.

encodeURI()

[JavaScriptMethodGlobalEncodeURI]

The JavaScript Global `encodeURI()` method encodes a string representing a URI by replacing certain characters by one, two, three or (rarely) four escape sequences representing the UTF-8 encoding of the character.

It has the following syntax with the following parameters:

`encodeURI(`*URI*`)`

Parameter	Required / Optional	Description
URI	Required	String representing URI to be encoded

More detail on URI encoding is given here.

encodeURIComponent()

[JavaScriptMethodGlobalEncodeURIComponent]

The JavaScript Global `encodeURIComponent()` method encodes a string representing a URI by replacing certain characters by one, two, three or (rarely) four escape sequences representing the UTF-8 encoding of the character.

It has the following syntax with the following parameters:

`encodeURIComponent(`*URI*`)`

Parameter	Required / Optional	Description
URI	Required	String representing URI to be encoded

More detail on URI encoding is given here.

escape()

[JavaScriptMethodGlobalEscape]

The JavaScript Global `escape()` method encodes a string representing a URI by replacing certain characters by one, two, three or (rarely) four escape sequences representing the UTF-8 encoding of the character. It is depreciated, and encodeURI or encodeURIComponent should be used instead.

It has the following syntax with the following parameters:

`escape(`*URI*`)`

Parameter	Required / Optional	Description
URI	Required	String representing URI to be encoded

More details on URI encoding is given here.

eval()

[JavaScriptMethodGlobalEval]

The JavaScript Global `eval()` method evaluates or executes an expression or set of JavaScript statements.

It has the following syntax with the following parameters:

`eval(`*str*`)`

Parameter	Required / Optional	Description
str	Required	String representing an expression or set of statements to evaluate / execute. If the argument is not a string, then it is left unchanged.

It is not appropriate to call `eval()` to evaluate an arithmetic expression, as JavaScript evaluates such expressions automatically.

`eval()` should be called sparingly. For example, it executes codes passed to it with the privileges of the calling algorithm, which can be exploited by a malicious party. It also invokes the JavaScript interpreter, which can frustrate modern browsers' ways of optimising code execution.

isFinite()

[JavaScriptMethodGlobalIsFinite]

The JavaScript Global `isFinite()` method indicates whether a number is a finite legal number. It returns `true` if the value is a finite number, otherwise returns `false`.

It has the following syntax with the following parameters:

`isFinite(`*x*`)`

Parameter	Required / Optional	Description
x	Required	Input parameter

The Number.isFinite method is subtly different to the global `isFinite` function. The latter coerces a value to a number before testing it, whilst the former does not. So, `Number.isFinite("4.3")` returns `false`, whilst `isFinite("4.3")` returns `true`.

isNaN()

[JavaScriptMethodGlobalIsNaN]

The JavaScript Global `isNaN()` method returns `true` if the value equates to NaN (after initial conversion to a Number) otherwise returns `false`.

It has the following syntax with the following parameters:

```
isNaN(x)
```

Parameter	Required / Optional	Description
x	Required	Input parameter

The Number.isNaN method is subtly different to the global `isNaN` function. The latter coerces a value to a number before testing it, whilst the former does not. So, `Number.isNaN("NaN")` returns `false`, whilst `isNaN("NaN")` returns `true`.

Number()

[JavaScriptMethodGlobalNumber]

The JavaScript Global `Number()` method converts an object to a number representing the value of the object (if this is possible) or returns `NaN` if such a conversion is not possible. If the parameter is a Date object then it returns the number of milliseconds since midnight 1 Jan 1970 (UTC).

It has the following syntax with the following parameters:

```
Number(x)
```

Parameter	Required / Optional	Description
x	Optional	Input parameter. If missing then returns 0.

parseFloat()

[JavaScriptMethodGlobalParseFloat]

The JavaScript Global `parseFloat()` method parses a string and returns a floating point number, assuming that the string can be interpreted as such a number. It finds the first character after leading spaces, works out if this is one that can appear in a number and then continues parsing the string until it reaches the end of any part that is interpretable as a number, returning that number as a number value, not a string. If the first available non-space character is not numerical then the method returns `NaN`. Only the first of multiple numbers will be returned.

It has the following syntax with the following parameters:

```
parseFloat(x)
```

Parameter	Required / Optional	Description
x	Required	String forming the input parameter

parseInt()

[JavaScriptMethodGlobalParseInt]

The JavaScript Global `parseInt()` method parses a string returns an integer where practical. If the first available non-space character of *x* is not numerical then the method returns `NaN`. Only the first of multiple numbers will be returned.

It has the following syntax with the following parameters:

`parseInt(`*x, radix*`)`

Parameter	Required / Optional	Description
x	Required	String corresponding to the input number
radix	Optional	An integer between 2 and 36 specifying the radix, i.e. number base, used in forming the integer from the strong. See below if omitted

Users are recommended to include a radix as the results can otherwise vary by browser. Usually, if the radix is omitted and *x* begins with "0x" (maybe "Ox") then the radix (number base) is taken to be 16. If it begins with "0" (maybe "O") then the radix is base 8 (this option is depreciated), otherwise usually the radix is defaulted to base 10 (i.e. decimal). The method finds the first character in *x* after leading spaces, works out if this is one that can appear in a number and then continues parsing the string until it reaches the end of any part that is interpretable as a number, returning that number as a number value, not a string.

String()

[JavaScriptMethodGlobalString]

The JavaScript Global `String()` method converts an object to a string. It returns the same as the `toString()` method of the object.

It has the following syntax with the following parameters:

`String(`*x*`)`

Parameter	Required / Optional	Description
x	Required	Input parameter.

unescape()

[JavaScriptMethodGlobalUnescape]

The JavaScript Global `unescape()` method inverts the outcome of encoding a string using encodeURI. It is depreciated, and decodeURI or decodeURIComponent should be used instead.

It has the following syntax with the following parameters:

`unescape(`*encodedURI*`)`

Parameter	Required / Optional	Description
encodedURI	Required	String representing an encoded URI

More details on URI encoding is given here.

HTML DOM media properties and methods

[HTMLDOMMediaPropertiesAndMethods]

The DOM objects corresponding to the HTML media elements (i.e. <audio> and <video> elements) both support a number of specific media-orientated properties and methods.

Media properties:

Property	Description	More
`audioTracks`	Returns AudioTrackList object indicating available audio tracks	Here
`buffered`	Returns TimeRanges object representing parts buffered	Here
`controller`	Returns current MediaController object for audio	Here
`crossOrigin`	Sets / returns CORS settings	Here
`currentSrc`	Returns media's URL	Here
`currentTime`	Sets / returns current playback position (in seconds)	Here
`defaultMuted`	Sets / returns if muted by default	Here
`defaultPlaybackRate`	Sets / returns default playback speed	Here
`duration`	Returns length of media (in seconds)	Here
`ended`	Returns whether playback has ended	Here
`error`	Returns MediaError object indicating error state of audio	Here
`muted`	Sets / returns whether sound muted	Here
`mediaGroup`	Sets / returns name of media group of which media is a part	Here
`networkState`	Returns current network state of media	Here
`paused`	Sets / returns whether media paused	Here
`playbackRate`	Sets / returns media playback speed	Here
`played`	Returns TimeRanges object representing parts played	Here
`readyState`	Returns current ready state	Here
`seekable`	Returns TimeRanges object representing seekable parts	Here
`seeking`	Returns whether user is currently seeking in media	Here
`textTracks`	Returns TextTrackList object indicating available text tracks	Here
`volume`	Sets / returns audio volume	Here

Media methods:

Method	Description	More
`addTextTrack()`	Adds new text track to media	Here
`canPlayType()`	Indicates if browser can play media type	Here
`load()`	Re-loads media	Here
`pause()`	Pauses media	Here
`play()`	Starts playing media	Here

audio methods (other than media methods):

fastSeek()

[JavaScriptMethodAudioFastSeek]

The `fastSeek()` method of the JavaScript DOM object corresponding to the HTML <audio> element seeks to a specified time in the audio.

getStartDate()

[JavaScriptMethodAudioGetStartDate]

The `getStartDate()` method of the JavaScript DOM object corresponding to the HTML <audio> element returns a Date object representing the current timeline offset.

canvas methods:

getContext()

[JavaScriptMethodCanvasGetContext]

The `getContext()` method of the JavaScript DOM object corresponding to the HTML <canvas> element returns an object that can be used to elaborate (populate) the canvas.

It has the following syntax with the following parameters. It returns a canvas context (or null if the context *type* is not recognised.

canvas.`getContext(`*type, attributes*`)`

Parameter	Required / Optional	Description
type	Required	String representing type of canvas drawing context. Possible values include: - `2d` - `webgl` - `webgl2` - `bitmaprenderer`
attributes	Optional	A set of additional attributes that can e.g. help the canvas software. Format is like `{alpha: false, … }`. Options vary depending on the context *type*. For `2d` the only universally recognised attribute is `alpha`, indicating if the canvas contains an alpha channel (so if false tells the browser that the backdrop is always opaque, which can speed up rendering). For `webgl`, some other attributes are acceptable

restore()

[JavaScriptMethodCanvasRestore]

The `restore()` method of the JavaScript DOM object corresponding to a context applied to the HTML <canvas> element returns the context's previously saved drawing state and attributes. Context states are stored on a stack every time the save() method is called, and returned whenever the `restore()` method is called. The `restore()` method takes no parameters.

For contexts generated by `getContext("2d")`, the state characteristics that are saved or restored appear to include:

- The current transformation matrix and clipping region
- The current values of the "Styles etc.", "Line styles", "Text" and "Other" properties of the context, see here

The current path is not part of the drawing state (it can only be reset using the beginPath method). Neither is the bitmap that has been drawn (it is a property of the canvas itself, not the context).

save()

[JavaScriptMethodCanvasSave]

The `save()` method of the JavaScript DOM object corresponding to a context applied to the HTML <canvas> element saves the context's drawing state and attributes. Context states are stored on a stack every time the `save()` method is called, and returned whenever the restore() method is called. The `save()` method takes no parameters.

For contexts generated by `getContext("2d")`, the state characteristics that are saved or restored appear to include:

- The current transformation matrix and clipping region
- The current values of the "Styles etc.", "Line styles", "Text" and "Other" properties of the context, see here

The current path is not part of the drawing state (it can only be reset using the beginPath method). Neither is the bitmap that has been drawn (it is a property of the canvas itself, not the context).

canvas2d properties:

data

[JavaScriptPropertyCanvas2dData]

The `data` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element returns an object containing image data for a specific ImageData object.

fillStyle

[JavaScriptPropertyCanvas2dFillStyle]

The `fillStyle` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the style (colour, gradient, pattern etc.) used to fill the element.

font

[JavaScriptPropertyCanvas2dFont]

The `font` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the CSS font property for the current text.

globalAlpha

[JavaScriptPropertyCanvas2dGlobalAlpha]

The `globalAlpha` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the current alpha, i.e. transparency value, of the drawing.

globalCompositeOperation

[JavaScriptPropertyCanvas2dGlobalCompositeOperation]

The `globalCompositeOperation` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns how new images are drawn onto existing images.

It can take the following values:

Value	Meaning
`copy`	Source image only (destination image is ignored)
`destination-atop`	As per source-atop but with source and destination flipped
`destination-in`	As per source-in but with source and destination flipped
`destination-out`	As per source-out but with source and destination flipped
`destination-over`	As per source-over but with source and destination flipped
`lighter`	Source image + destination image
`source-atop`	Source image on top of destination image (part of source image outside destination image is ignored)
`source-in`	Source image into destination image (only part of source image inside destination image is shown, destination image is transparent)
`source-out`	Source image out of destination image (only part of source image outside destination image is shown, destination image is transparent)
`source-over`	(default). Source image over destination image
`xor`	Source and destination images combined using XOR operation

height

[JavaScriptPropertyCanvas2dHeight]

The `height` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element returns the height of an ImageData object.

lineCap

[JavaScriptPropertyCanvas2dLineCap]

The `lineCap` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the style used for line ends.

It can take the following values: `butt` (default, a flat edge), `round` (rounded end cap) or `square` (square end cap).

lineJoin

[JavaScriptPropertyCanvas2dLineJoin]

The `lineJoin` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the type of corner between two lines where they join.

It can take the following values: `bevel` (creates a bevelled corner), `round` (creates a rounded corner) or `miter` (default, creates a sharp corner, provided the distance between the inner and outer corner of the join is not larger than the miterLimit).

lineWidth

[JavaScriptPropertyCanvas2dLineWidth]

The `lineWidth` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the line width.

miterLimit

[JavaScriptPropertyCanvas2dMiterLimit]

The `miterLimit` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the maximum mitre limit.

The mitre is the distance between the inner and outer corner where two lines meet. The `miterLimit` property is only relevant if the lineJoin property is `miter`. It will apply when the angle between the two lines is small, when the corner will be displayed as if its lineJoin property is `bevel`.

shadowBlur

[JavaScriptPropertyCanvas2dShadowBlur]

The `shadowBlur` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the shadow blur level, see CSS text-shadow property.

shadowColor

[JavaScriptPropertyCanvas2dShadowColor]

The `shadowColor` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the shadow colour, see CSS text-shadow property.

shadowOffsetX

[JavaScriptPropertyCanvas2dShadowOffsetX]

The `shadowOffsetX` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the shadow horizontal offset, see CSS text-shadow property.

shadowOffsetY

[JavaScriptPropertyCanvas2dShadowOffsetY]

The `shadowOffsetY` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the shadow vertical offset, see CSS text-shadow property.

strokeStyle

[JavaScriptPropertyCanvas2dStrokeStyle]

The `strokeStyle` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the style used for strokes.

textAlign

[JavaScriptPropertyCanvas2dTextAlign]

The `textAlign` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the CSS text-align property for the current text.

textBaseline

[JavaScriptPropertyCanvas2dTextBaseline]

The `textBaseline` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element sets / returns the text baseline for current text.

It can take the following values: `alphabetic` (default), `bottom`, `hanging`, `ideographic`, `middle`, `top`. These are not always interpreted in the same manner in all browsers.

width

[JavaScriptPropertyCanvas2dWidth]

The `width` property of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element returns the width of an ImageData object.

canvas2d methods:

addColorStop()

[JavaScriptMethodCanvas2dAddColorStop]

The `addColorStop()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element specifies colours and stop positions for a gradient object, created by createLinearGradient() or createRadialGradient(). You need to include at least one colour stop for a gradient to be visible.

It has the following syntax with the following parameters.

gradient`.addColorStop(`*stop, color*`)`

Parameter	Required / Optional	Description
type	Required	A value between 0.0 and 1.0 identifying the position of the stop used in a gradient
color	Optional	Specified CSS colour to display at the position of the *stop*

arc()

[JavaScriptMethodCanvas2dArc]

The `arc()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a circular arc.

It has the following syntax with the following parameters.

context`.arc(`*x, y, r, startAngle, endAngle, counterclockwise*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate or circle centre
y	Required	y-coordinate or circle centre
r	Required	Radius of circle
startAngle	Required	Start angle of arc in radians from x-axis

endAngle	Required	End angle of arc in radians from x-axis
counterclockwise	Optional	(default is false). Boolean, if true then draw arc counterclockwise, otherwise clockwise

arcTo()

[JavaScriptMethodCanvas2dArcTo]

The `arcTo()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a circular arc between two tangents.

It has the following syntax with the following parameters.

context.`arcTo(`*x, y, r, startAngle, endAngle, counterclockwise*`)`

Parameter	Required / Optional	Description
xstart	Required	x-coordinate of start tangent
ystart	Required	y-coordinate of start tangent
xend	Required	x-coordinate of end tangent
yend	Required	y-coordinate of end tangent
r	Required	Radius of arc

beginPath()

[JavaScriptMethodCanvas2dBeginPath]

The `beginPath()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element begins / resets a path. The stroke() method actually draws the path on the canvas.

It has the following syntax (with no parameters).

context.`beginPath()`

bezierCurveTo()

[JavaScriptMethodCanvas2dBezierCurveTo]

The `bezierCurveTo()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a cubic Bézier curve. To create a quadratic Bézier curve use the quadraticCurveTo() method.

It has the following syntax with the following parameters.

context.`bezierCurveTo(`*x, y, r, startAngle, endAngle, counterclockwise*`)`

Parameter	Required / Optional	Description
pt1x	Required	x-coordinate of first control point of curve

pt1y	Required	y-coordinate of first control point of curve
pt2x	Required	x-coordinate of second control point of curve
pt2y	Required	y-coordinate of second control point of curve
x	Required	x-coordinate of end point
y	Required	y-coordinate of end point

clearRect()

[JavaScriptMethodCanvas2dClearRect]

The `clearRect()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element clears specified pixels within a rectangle.

It has the following syntax with the following parameters.

context.`clearRect(`*x, y, width, height*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of upper-left corner
y	Required	y-coordinate of upper-left corner
width	Required	Width of rectangle, in pixels
height	Required	Height of rectangle, in pixels

clip()

[JavaScriptMethodCanvas2dClip]

The `clip()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element clips a region from canvas. Once a region is clipped, all future drawing is limited to the clipped region, although if you save the region before clipping it can then be restored using the restore() method.

It has the following syntax with no parameters.

context.`clip()`

closePath()

[JavaScriptMethodCanvas2dClosePath]

The `closePath()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element completes a path back to its original starting point.

It has the following syntax with no parameters.

context.`closePath()`

createImageData()

[JavaScriptMethodCanvas2dCreateImageData]

The `createImageData()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a new blank ImageData object.

Each pixel in the ImageData has 4 values, i.e. its RGBA values (see CSS Colours). The data is held in an array which is 4 times the size of the ImageData object, i.e. width x height x 4. This is stored in the `data` property of the ImageDataObject.

There are two versions of the `createImageData()` method with the following formats and parameters:

context.`createImageData(`*width, height*`)`

Parameter	Required / Optional	Description
width	Required	Width of ImageData, in pixels
height	Required	Height of ImageData, in pixels

context.`createImageData(`*imageData*`)`

Parameter	Required / Optional	Description
imageData	Required	ImageData object to be used as a template for the new object (note only dimensions are used, the image data is not copied

createLinearGradient()

[JavaScriptMethodCanvas2dCreateLinearGradient]

The `createLinearGradient()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a linear gradient. The resulting object can be used as the value of the strokeStyle or fillStyle properties and hence to fill in rectangles, circles etc. You need to apply some addColorStop() methods to the gradient for it to be visible.

It has the following syntax with the following parameters.

context.`createLinearGradient(`*xstart, ystart, xend, yend*`)`

Parameter	Required / Optional	Description
xstart	Required	x-coordinate of gradient start point
xend	Required	y-coordinate of gradient start point
ystart	Required	x-coordinate of gradient end point
yend	Required	y-coordinate of gradient end point

createPattern()

[JavaScriptMethodCanvas2dCreatePattern]

The `createPattern()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element repeats a specific element in a specific direction (the element can be an image, video or another `canvas` element).

It has the following syntax with the following parameters.

context.`createPattern(`*image, repeatspecification*`)`

Parameter	Required / Optional	Description
image	Required	Element to be used in pattern
repeatspecification	Required	Parameter that specifies how repeating is to be carried out. Can take one of these values: - `repeat`: pattern repeats both horizontally and vertically - `repeat-x`: pattern only repeats horizontally - `repeat-y`: pattern only repeats vertically - `no-repeat`: pattern not repeated

createRadialGradient()

[JavaScriptMethodCanvas2dCreateRadialGradient]

The `createRadialGradient()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a radial (i.e. circular) gradient. The resulting object can be used as the value of the strokeStyle or fillStyle properties and hence to fill in rectangles, circles etc. You need to apply some addColorStop() methods to the gradient for it to be visible.

It has the following syntax with the following parameters.

context.`createRadialGradient(`*xstart, ystart, rstart, xend, yend, rend*`)`

Parameter	Required / Optional	Description
xstart	Required	x-coordinate of centre of starting circle of gradient
xend	Required	y-coordinate of centre of starting circle of gradient
rstart	Required	Radius of starting circle
ystart	Required	x-coordinate of centre of ending circle of gradient
yend	Required	y-coordinate of centre of ending circle of gradient
rend	Required	Radius of ending circle

drawImage()

[JavaScriptMethodCanvas2dDrawImage]

The `drawImage()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element draws an image, canvas or video onto the canvas.

There are three versions of the `createImageData()` method with the following formats and parameters (the ones with clip parameters involve pre-clipping of the image):

context.`drawImage(`*image, x, y*`)`

context.`drawImage(`*image, x, y, width, height*`)`

context.`drawImage(`*image, clipx, clipy, clipwidth, clipheight, x, y, width, height*`)`

Parameter	Required / Optional	Description
clipheight	Optional	Clip height
clipwidth	Optional	Clip width
clipx	Optional	Clip x-coordinate
clipy	Optional	Clip y-coordinate
height	Optional	Height of image to use (potentially stretching or squashing original image)
image	Required	Image / video / canvas to be inserted
width	Optional	Width of image to use (potentially stretching or squashing original image)
x	Required	x-coordinate of upper-left corner
y	Required	y-coordinate of upper-left corner

fill()

[JavaScriptMethodCanvas2dFill]

The `fill()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element fills the current path. If the path is not closed then this method will add a line from the last point to the start point of the path, like closePath().

It has the following syntax with no parameters.

context.`fill()`

fillRect()

[JavaScriptMethodCanvas2dFillRect]

The `fillRect()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element draws a 'filled' rectangle.

It has the following syntax with the following parameters.

context.`fillRect(`*x, y, width, height*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of upper-left corner
y	Required	y-coordinate of upper-left corner
width	Required	Width of rectangle, in pixels
height	Required	Height of rectangle, in pixels

fillText()

[JavaScriptMethodCanvas2dFillText]

The `fillText()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element draws 'filled' text.

It has the following syntax with the following parameters.

context.`fillText(`*text, x, y, maxwidth*`)`

Parameter	Required / Optional	Description
text	Required	String specifying text
x	Required	x-coordinate of upper-left corner (relative to canvas)
y	Required	y-coordinate of upper-left corner (relative to canvas)
maxwidth	Optional	Maximum width, in pixels

getImageData()

[JavaScriptMethodCanvas2dGetImageData]

The `getImageData()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element returns an ImageData object characterised by the pixel data for a specific rectangle in the canvas.

Each pixel in the ImageData has 4 values, i.e. its RGBA values (see CSS Colours). The data is held in an array which is 4 times the size of the ImageData object, i.e. width x height x 4. This is stored in the `data` property of the ImageDataObject.

It has the following syntax with the following parameters.

context.`getImageData(`*x, y, width, height*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of upper-left corner
y	Required	y-coordinate of upper-left corner
width	Required	Width of rectangle, in pixels
height	Required	Height of rectangle, in pixels

isPointInPath()

[JavaScriptMethodCanvas2dIsPointInPath]

The `isPointInPath()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element returns true if point is in current path, otherwise false.

It has the following syntax with the following parameters.

context.`isPointInPath(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of point
y	Required	y-coordinate of point

lineTo()

[JavaScriptMethodCanvas2dLineTo]

The `lineTo()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element moves the path to a specified point in the canvas, creating a line from the previous point.

It has the following syntax with the following parameters.

context.`lineTo(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of point
y	Required	y-coordinate of point

measureText()

[JavaScriptMethodCanvas2dMeasureText]

The `measureText()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element returns an object indicating the width of the specified text. It can be used to measure the width of some text before it is written onto the canvas

The width can be found using the following syntax with the following parameters.

context.`measureText(`*text*`).width`

Parameter	Required / Optional	Description
text	Required	String specifying text

moveTo()

[JavaScriptMethodCanvas2dMoveTo]

The `moveTo()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element moves the path to a specified point in the canvas, without creating a line from the previous point.

It has the following syntax with the following parameters.

context.`moveTo(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of point
y	Required	y-coordinate of point

putImageData()

[JavaScriptMethodCanvas2dPutImageData]

The `putImageData()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element puts image data included in an ImageData object onto the canvas.

It has the following syntax with the following parameters.

context.`putImageData(`*ximgData, x, y, drawnx,drawny, drawnwidth, drawnheight*`)`

Parameter	Required / Optional	Description
imgData	Required	ImageData object to be inserted back onto canvas
x	Required	x-coordinate of upper-left corner of ImageData object
y	Required	y-coordinate of upper-left corner of ImageData object
drawnx	Optional	x-coordinate of upper-left corner of rectangle drawn onto canvas
drawny	Optional	x-coordinate of upper-left corner of rectangle drawn onto canvas
drawnwidth	Optional	Width of rectangle, in pixels, drawn onto canvas
drawnheight	Optional	Height of rectangle, in pixels, drawn onto canvas

quadraticCurveTo()

[JavaScriptMethodCanvas2dQuadraticCurveTo]

The `quadraticCurveTo()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a quadratic Bézier curve. To create a cubic Bézier curve use the bezierCurveTo() method.

It has the following syntax with the following parameters.

context.`quadraticCurveTo(`*x, y, r, startAngle, endAngle, counterclockwise*`)`

Parameter	Required / Optional	Description
ptx	Required	x-coordinate of control point of curve
pty	Required	y-coordinate of control point of curve
x	Required	x-coordinate of end point
y	Required	y-coordinate of end point

rect()

[JavaScriptMethodCanvas2dRect]

The `rect()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element creates a rectangle. Use the stroke() or fill() methods to draw the rectangle on the canvas.

It has the following syntax with the following parameters.

context.`rect(`*x, y, width, height*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of upper-left corner
y	Required	y-coordinate of upper-left corner
width	Required	Width of rectangle, in pixels
height	Required	Height of rectangle, in pixels

rotate()

[JavaScriptMethodCanvas2dRotate]

The `rotate()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element rotates the current drawing. It only affects drawings made after the rotation is applied

It has the following syntax with the following parameters.

context.`rotate(`*angle*`)`

Parameter	Required / Optional	Description
angle	Required	Angle in radians

scale()

[JavaScriptMethodCanvas2dScale]

The `scale()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element scales the current drawing. It also scales future drawings and the positioning is also scaled. The parameters are scaling factors, so 1 means stay at 100% of previous size, 0.5 means adjust to 50% of previous size etc.

It has the following syntax with the following parameters:

context`.scale(`*scalewidth*`,` *scaleheight*`)`

Parameter	Required / Optional	Description
scalewidth	Required	Scaling factor applied to width
scaleheight	Required	Scale factor applied to height (

setTransform()

[JavaScriptMethodCanvas2dSetTransform]

The `setTransform()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element defines a transform matrix and then applies the transform() method.

It has the following syntax with the following parameters:

context`.setTransform(`*x1*`,` *x2*`,` *x3*`,` *x4, x5, x6*`)`

Parameter	Required / Optional	Description
x1	Required	Horizontal scaling
x2	Required	Horizontal skew
x3	Required	Vertical skew
x4	Required	Vertical scaling
x5	Required	Horizontal moving
x6	Required	Vertical moving

stroke()

[JavaScriptMethodCanvas2dStroke]

The `stroke()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element draws a path in the canvas. The default colour is black, but this can be overridden using the strokeStyle property.

It has the following syntax with no parameters.

context`.stroke()`

strokeRect()

[JavaScriptMethodCanvas2dStrokeRect]

The `strokeRect()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element draws a rectangle that is not 'filled' (i.e. it only draws the edge of the rectangle). The default colour is black, but can be overridden using the strokeStyle property.

It has the following syntax with the following parameters.

context.`strokeRect(`*x, y, width, height*`)`

Parameter	Required / Optional	Description
x	Required	x-coordinate of upper-left corner
y	Required	y-coordinate of upper-left corner
width	Required	Width of rectangle, in pixels
height	Required	Height of rectangle, in pixels

strokeText()

[JavaScriptMethodCanvas2dStrokeText]

The `strokeText()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element draws text that is not 'filled' (i.e. it only draws the edges of the characters). The default colour is black, but can be overridden using the strokeStyle property.

It has the following syntax with the following parameters.

context.`strokeText(`*text, x, y, maxwidth*`)`

Parameter	Required / Optional	Description
text	Required	String specifying text
x	Required	x-coordinate of upper-left corner (relative to canvas)
y	Required	y-coordinate of upper-left corner (relative to canvas)
maxwidth	Optional	Maximum width, in pixels

transform()

[JavaScriptMethodCanvas2dTransform]

The `transform()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element applies a transformation to the current drawing.

It has the following syntax with the following parameters:

context`.transform(`*x1*`,`*x2*`,`*x3*`,`*x4, x5, x6*`)`

Parameter	Required / Optional	Description
x1	Required	Horizontal scaling
x2	Required	Horizontal skew
x3	Required	Vertical skew
x4	Required	Vertical scaling
x5	Required	Horizontal moving
x6	Required	Vertical moving

translate()

[JavaScriptMethodCanvas2dTranslate]

The `translate()` method of the JavaScript DOM object returned by the `getContext("2d")` method applied to the HTML <canvas> element applies a translation to current drawing, i.e. adjusts the position of its origin, remapping the position of the coordinate (0,0).

It has the following syntax with the following parameters.

context`.translate(`*x, y*`)`

Parameter	Required / Optional	Description
x	Required	Value added to x-coordinates
y	Required	Value added to y-coordinates

datalist properties:

options

[JavaScriptPropertyDatalistOptions]

The `options` property of the JavaScript DOM object corresponding to the HTML <datalist> element returns a collection of all options included in the <datalist>.

fieldset properties:

type

[JavaScriptPropertyFieldsetType]

The `type` property of the JavaScript DOM object corresponding to the HTML <fieldset> element returns the type of the form element that the <fieldset> element belongs to.

form properties:

encoding

[JavaScriptPropertyFormEncoding]

The `encoding` property of the JavaScript DOM object corresponding to the HTML <form> element is an alias for its enctype property.

length

[JavaScriptPropertyFormLength]

The `length` property of the JavaScript DOM object corresponding to the HTML <form> element returns the number of elements in the <form>.

form methods:

reset()

[JavaScriptMethodFormReset]

The `reset()` method of the JavaScript DOM object corresponding to the HTML <form> element resets the <form>.

It has the following syntax with no parameters.

formObject.`reset()`

submit()

[JavaScriptMethodFormSubmit]

The `submit()` method of the JavaScript DOM object corresponding to the HTML <form> element submits the <form>.

It has the following syntax with no parameters.

formObject.`submit()`

iframe properties:

contentDocument

[JavaScriptPropertyIframeContentDocument]

The `contentDocument` property of the JavaScript DOM object corresponding to the HTML <iframe> element returns the document object generated by the <iframe>.

contentWindow

[JavaScriptPropertyIframeContentWindow]

The `contentWindow` property of the JavaScript DOM object corresponding to the HTML <iframe> element returns the window object generated by the <iframe>.

img properties:

complete

[JavaScriptPropertyImgComplete]

The `complete` property of the JavaScript DOM object corresponding to the HTML <img> element returns whether the browser has finished loading the image underlying the <img> element.

naturalHeight

[JavaScriptPropertyImgNaturalHeight]

The `naturalHeight` property of the JavaScript DOM object corresponding to the HTML <img> element returns the original height of the image underlying the <img> element.

naturalWidth

[JavaScriptPropertyImgNaturalWidth]

The `naturalWidth` property of the JavaScript DOM object corresponding to the HTML <img> element returns the original width of the image underlying the <img> element.

input properties:

defaultChecked

[JavaScriptPropertyInputDefaultChecked]

The `defaultChecked` property of the JavaScript DOM object corresponding to the HTML <input> element (of type `checkbox` or `radio`) returns the default value of the checked attribute of the <input> element.

defaultValue

[JavaScriptPropertyInputDefaultValue]

The `defaultValue` property of the JavaScript DOM object corresponding to the HTML <input> element sets / returns the default value.

files

[JavaScriptPropertyInputFiles]

The `files` property of the JavaScript DOM object corresponding to the HTML <input> element (of type `file`) returns a FileList object representing file(s) selected by upload button.

form

[JavaScriptPropertyInputForm]

The `form` property of the JavaScript DOM object corresponding to the HTML <input> element returns the form that contains the element.

indeterminate

[JavaScriptPropertyInputIndeterminate]

The `indeterminate` property of the JavaScript DOM object corresponding to the HTML <input> element (of type `checkbox`) sets / returns the value of its indeterminate status.

input methods:

select()

[JavaScriptMethodInputSelect]

The `select()` method of the JavaScript DOM object corresponding to the HTML <input> element selects the (text) content of the field.

It has the following syntax with no parameters.

textObject.`select()`

stepDown()

[JavaScriptMethodInputStepDown]

The `stepDown()` method of the JavaScript DOM object corresponding to the HTML <input> element (of type `datetime`, `datetime-local`, `month`, `number`, `range`, `time`, `week`) decrements the value of the relevant field by a specified amount.

It has the following syntax with the following parameters.

object.`stepDown(`*x*`)`

Parameter	Required / Optional	Description
x	Optional	(default is 1). Specifies amount by which number field value is decreased

stepUp()

[JavaScriptMethodInputStepUp]

The `stepUp()` method of the JavaScript DOM object corresponding to the HTML <input> element (of type `datetime`, `datetime-local`, `month`, `number`, `range`, `time`, `week`) increments the value of the relevant field by a specified amount.

It has the following syntax with the following parameters.

object.`stepUp(`*x*`)`

Parameter	Required / Optional	Description
x	Optional	(default is 1). Specifies amount by which number field value is increased

keygen properties:

type

[JavaScriptPropertyKeygenType]

The `type` property of the JavaScript DOM object corresponding to the HTML <keygen> element returns the type of the form element in which the keygen field appears.

legend properties:

form

[JavaScriptPropertyLegendForm]

The `form` property of the JavaScript DOM object corresponding to the HTML <legend> element returns the form that contains the element.

map properties:

areas

[JavaScriptPropertyMapAreas]

The `areas` property of the JavaScript DOM object corresponding to the HTML <map> element returns a collection of all <area> elements linked to the <map> element.

images

[JavaScriptPropertyMapImages]

The `images` property of the JavaScript DOM object corresponding to the HTML <map> element returns a collection of all <img> and <object> elements linked to the <map> element.

media properties:

audioTracks

[JavaScriptPropertyMediaAudioTracks]

The `audioTracks` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns an AudioTrackList object indicating available audio tracks.

buffered

[JavaScriptPropertyMediaBuffered]

The `buffered` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns a TimeRanges object representing the parts that are buffered.

controller

[JavaScriptPropertyMediaController]

The `controller` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns the current MediaController object for audio.

crossOrigin

[JavaScriptPropertyMediaCrossOrigin]

The `crossOrigin` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns CORS settings of the element.

currentSrc

[JavaScriptPropertyMediaCurrentSrc]

The `currentSrc` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns the media's URL.

currentTime

[JavaScriptPropertyMediaCurrentTime]

The `currentTime` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns current playback position (in seconds).

defaultMuted

[JavaScriptPropertyMediaDefaultMuted]

The `defaultMuted` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns if muted by default.

defaultPlaybackRate

[JavaScriptPropertyMediaDefaultPlaybackRate]

The `defaultPlaybackRate` property of the JavaScript DOM object corresponding HTML media elements (i.e. <audio> and <video> elements) sets / returns the default playback speed.

duration

[JavaScriptPropertyMediaDuration]

The `duration` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns the length of media (in seconds).

ended

[JavaScriptPropertyMediaEnded]

The `ended` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns whether playback has ended.

error

[JavaScriptPropertyMediaError]

The `ended` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns whether playback has ended.

mediaGroup

[JavaScriptPropertyMediaMediaGroup]

The `mediaGroup` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns name of media group of which the media is a part.

muted

[JavaScriptPropertyMediaMuted]

The `muted` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns whether sound muted.

networkState

[JavaScriptPropertyMediaNetworkState]

The `networkState` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns the current network state of the media.

paused

[JavaScriptPropertyMediaPaused]

The `paused` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns whether the media is paused.

playbackRate

[JavaScriptPropertyMediaPlaybackRate]

The `playbackRate` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns the media playback speed.

played

[JavaScriptPropertyMediaPlayed]

The `played` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns a TimeRanges object representing the parts played.

readyState

[JavaScriptPropertyMediaReadyState]

The `readyState` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns the current ready state.

seekable

[JavaScriptPropertyMediaSeekable]

The `seekable` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns a TimeRanges object representing the seekable parts of the media.

seeking

[JavaScriptPropertyMediaSeeking]

The `seeking` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns whether user is currently seeking in the media.

textTracks

[JavaScriptPropertyMediaTextTracks]

The `textTracks` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) returns a TextTrackList object indicating available text tracks.

volume

[JavaScriptPropertyMediaVolume]

The `volume` property of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) sets / returns the audio volume.

media methods:

addTextTrack()

[JavaScriptMethodMediaAddTextTrack]

The `addTextTrack()` method of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) adds a new text track to the media. It is not currently supported by many major browsers.

It has the following syntax with the following parameters.

object.`addTextTrack(`*kind, label, language*`)`

Parameter	Required / Optional	Description
kind	Required	Kind of track being added. Possible values include: - `subtitles` - `caption` - `descriptions` - `chapters` - `metadata`
label	Optional	String specifying track label for users
language	Optional	Two-letter language code

canPlayType()

[JavaScriptMethodMediaCanPlayType]

The `canPlayType()` method of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) indicates if browser can play the media type.

It has the following syntax with the following parameters. It returns a string indicating likely level of support. Possible return values include: “probably” (most likely to support), “maybe” (might support) or “” (empty string, no support)

object.`canPlayType(`*type*`)`

Parameter	Required /	Description

	Optional	
type	Required	Type (and optional codecs) to test support for, e.g. `audio/mp4; codecs="mp4a40.5"` or `video/ogg`

load()

[JavaScriptMethodMediaLoad]

The `load()` method of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) loads / re-loads the media.

It has the following syntax with no parameters.

object.`load()`

pause()

[JavaScriptMethodMediaPause]

The `pause()` method of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) pauses the media.

It has the following syntax with no parameters.

object.`pause()`

play()

[JavaScriptMethodMediaPlay]

The `play()` method of the JavaScript DOM object corresponding to HTML media elements (i.e. <audio> and <video> elements) starts playing the media.

It has the following syntax with no parameters.

object.`play()`

menuitem properties:

command

[JavaScriptPropertyMenuitemCommand]

The `command` property of the JavaScript DOM object corresponding to the HTML <menuitem> element sets/ returns the command property of the DOM object.

meter properties:

labels
[JavaScriptPropertyMeterLabels]

The `labels` property of the JavaScript DOM object corresponding to the HTML <meter> element returns a collection of <label> elements corresponding to the labels used in the gauge (meter).

option properties:

defaultSelected
[JavaScriptPropertyOptionDefaultSelected]

The `defaultSelected` property of the JavaScript DOM object corresponding to the HTML <option> element returns the default value of the selected attribute.

form
[JavaScriptPropertyOptionForm]

The `form` property of the JavaScript DOM object corresponding to the HTML <option> element returns a reference to form that contains the option.

index
[JavaScriptPropertyOptionIndex]

The `index` property of the JavaScript DOM object corresponding to the HTML <option> element sets / returns the index position of an option in a drop-down list.

text
[JavaScriptPropertyOptionText]

The `text` property of the JavaScript DOM object corresponding to the HTML <option> element sets / returns the text of the option.

output properties:

defaultValue
[JavaScriptPropertyOutputDefaultValue]

The `defaultValue` property of the JavaScript DOM object corresponding to the HTML <output> element sets / returns the default value.

labels
[JavaScriptPropertyOutputLabels]

The `labels` property of the JavaScript DOM object corresponding to the HTML <output> element returns a collection of <label> elements associated with the <output> object.

type

[JavaScriptPropertyOutputType]

The `type` property of the JavaScript DOM object corresponding to the HTML <output> element returns the type of the HTML element represented by the <output> object.

value

[JavaScriptPropertyOutputValue]

The `value` property of the JavaScript DOM object corresponding to the HTML <output> element returns the value of the element.

progress properties:

labels

[JavaScriptPropertyProgressLabels]

The `labels` property of the JavaScript DOM object corresponding to the HTML <progress> element returns a list of the progress bar labels (if any).

position

[JavaScriptPropertyProgressPosition]

The `position` property of the JavaScript DOM object corresponding to the HTML <progress> element returns the current position of progress bar.

script properties:

crossOrigin

[JavaScriptPropertyScriptCrossOrigin]

The `crossOrigin` property of the JavaScript DOM object corresponding to the HTML <script> element sets / returns the CORS settings for the script.

text

[JavaScriptPropertyScriptText]

The `text` property of the JavaScript DOM object corresponding to the HTML <script> element sets / returns the contents of all child text nodes of the script.

select properties:

length

[JavaScriptPropertySelectLength]

The `length` property of the JavaScript DOM object corresponding to the HTML <select> element returns the number of option elements within the drop-down list.

options

[JavaScriptPropertySelectOptions]

The `options` property of the JavaScript DOM object corresponding to the HTML <select> element returns a collection of all options in drop-down list.

selectedIndex

[JavaScriptPropertySelectSelectedIndex]

The `selectedIndex` property of the JavaScript DOM object corresponding to the HTML <select> element sets / returns the index of the selected option.

type

[JavaScriptPropertySelectType]

The `type` property of the JavaScript DOM object corresponding to the HTML <select> element returns the type of form the drop-down list is within.

value

[JavaScriptPropertySelectValue]

The `value` property of the JavaScript DOM object corresponding to the HTML <select> element sets / returns the value of the selected option in the drop-down list.

select methods:

add()

[JavaScriptMethodSelectAdd]

The `add()` method of the JavaScript DOM object corresponding to the HTML <select> element adds an option to a drop-down list.

It has the following syntax with the following parameters:

object`.add(`*option, index*`)`

Parameter	Required / Optional	Description
option	Required	Option to be added (needs to be an <option> or an <optgroup> element)
index	Optional	(default is 0). Index position where new option element is inserted

remove()

[JavaScriptMethodSelectRemove]

The `remove()` method of the JavaScript DOM object corresponding to the HTML <select> element removes an option from a drop-down list.

It has the following syntax with the following parameters:

object.`remove`(*index*)

Parameter	Required / Optional	Description
index	Required	Index position from where option element is removed (index starts at 0)

table properties:

caption

[JavaScriptPropertyTableCaption]

The `caption` property of the JavaScript DOM object corresponding to the HTML <table> element returns the <caption> element of the table.

rows

[JavaScriptPropertyTableRows]

The `rows` property of the JavaScript DOM object corresponding to the HTML <table> element returns a collection of the <tr> elements of the table.

tBodies

[JavaScriptPropertyTableTBodies]

The `tBodies` property of the JavaScript DOM object corresponding to the HTML <table> element returns a collection of <tbody> elements of the table. <tbody> DOM elements have row-orientated JavaScript properties and methods like those of <table> elements.

tFoot

[JavaScriptPropertyTableTFoot]

The `tFoot` property of the JavaScript DOM object corresponding to the HTML <table> element returns the <tfoot> element of the table. <tfoot> DOM elements have row-orientated JavaScript properties and methods like those of <table> elements.

tHead

[JavaScriptPropertyTableTHead]

The `tHead` property of the JavaScript DOM object corresponding to the HTML <table> element returns the <thead> element of the table. <thead> DOM elements have row-orientated JavaScript properties and methods like those of <table> elements.

table methods:

createCaption()

[JavaScriptMethodTableCreateCaption]

The `createCaption()` method of the JavaScript DOM object corresponding to the HTML <table> element creates an empty <caption> element and adds it to the table. If a <caption> element already exists in the table then it returns the existing one, without creating a new one.

It has the following syntax with no parameters:

object`.createCaption()`

createTFoot()

[JavaScriptMethodTableCreateTFoot]

The `createTFoot()` method of the JavaScript DOM object corresponding to the HTML <table> element creates an empty <tfoot> element and adds it to the table. If a <tfoot> element already exists in the table then it returns the existing one, without creating a new one.

It has the following syntax with no parameters:

object`.createTFoot()`

createTHead()

[JavaScriptMethodTableCreateTHead]

The `createTHead()` method of the JavaScript DOM object corresponding to the HTML <table> element creates an empty <thead> element and adds it to the table. If a <thead> element already exists in the table then it returns the existing one, without creating a new one.

It has the following syntax with no parameters:

object.`createTHead()`

deleteCaption()

[JavaScriptMethodTableDeleteCaption]

The `deleteCaption()` method of the JavaScript DOM object corresponding to the HTML <table> element removes the first <caption> element from the table.

It has the following syntax with no parameters:

object.`deleteCaption()`

deleteRow()

[JavaScriptMethodTableDeleteRow]

The `deleteRow()` method of the JavaScript DOM object corresponding to the HTML <table> element removes a <tr> element from the table.

It has the following syntax with the following parameters:

object.`deleteRow(`*index*`)`

Parameter	Required / Optional	Description
index	Required (see below)	Index position from where row is removed (index starts at 0). Using zero deletes the first row, using -1 deletes the last row.

Note: the index parameter is required by some browsers but optional for others.

deleteTFoot()

[JavaScriptMethodTableDeleteTFoot]

The `deleteTFoot()` method of the JavaScript DOM object corresponding to the HTML <table> element removes the first <tfoot> element from the table.

It has the following syntax with no parameters:

object.`deleteTFoot()`

deleteTHead()

[JavaScriptMethodTableDeleteTHead]

The `deleteTHead()` method of the JavaScript DOM object corresponding to the HTML <table> element removes the <thead> element from the table.

It has the following syntax with no parameters:

object.`deleteTHead()`

insertRow()

[JavaScriptMethodTableInsertRow]

The `insertRow()` method of the JavaScript DOM object corresponding to the HTML <table> element creates an empty <tr> element and adds it to the table.

It has the following syntax with the following parameters:

object.`insertRow(`*index*`)`

Parameter	Required / Optional	Description
index	Required (see below)	Index position where row is to be inserted (index starts at 0). Using zero sometimes inserts a new row at the start and sometimes at the end depending on browser. Using -1 inserts after the end of the existing last row.

Note: the index parameter is required by some browsers but optional for others.

textarea properties:

defaultValue

[JavaScriptPropertyTextareaDefaultValue]

The `defaultValue` property of the JavaScript DOM object corresponding to the HTML <textarea> element sets / returns the default value of the element.

type

[JavaScriptPropertyTextareaType]

The `type` property of the JavaScript DOM object corresponding to the HTML <textarea> element returns the type of form that contains the element.

value

[JavaScriptPropertyTextareaValue]

The `value` property of the JavaScript DOM object corresponding to the HTML <textarea> element sets / returns the contents of the element.

textarea methods:

select()

[JavaScriptMethodTextareaSelect]

The `select()` method of the JavaScript DOM object corresponding to the HTML <textarea> element selects the entire contents of the text area.

It has the following syntax with no parameters:

object.`select()`

title properties:

text

[JavaScriptPropertyTitleText]

The `text` property of the JavaScript DOM object corresponding to the HTML <title> element sets / returns the text of the document title.

tr properties:

cells

[JavaScriptPropertyTrCells]

The `cells` property of the JavaScript DOM object corresponding to the HTML <tr> element sets / returns a collection of all the <td> and <th> elements in a row.

rowIndex

[JavaScriptPropertyTrRowIndex]

The `rowIndex` property of the JavaScript DOM object corresponding to the HTML <tr> element sets / returns the position of a row in the rows collection of a <table> element.

sectionRowIndex

[JavaScriptPropertyTrSectionRowIndex]

The `sectionRowIndex` property of the JavaScript DOM object corresponding to the HTML <tr> element sets / returns the position of a row in the rows collection of a <tbody>, <tfoot> or a <thead>.

tr methods:

deleteCell()

[JavaScriptMethodTrDeleteCell]

The `deleteCell()` method of the JavaScript DOM object corresponding to the HTML <tr> element deletes a cell from a table row.

It has the following syntax with the following parameters:

object.`deleteCell(`*index*`)`

Parameter	Required / Optional	Description
index	Required (see below)	Index position from where cell is removed (index starts at 0). Using zero deletes the first cell, using -1 deletes the last cell.

Note: the index parameter is required by some browsers but optional for others.

insertCell()

[JavaScriptMethodTrInsertCell]

The `insertCell()` method of the JavaScript DOM object corresponding to the HTML <tr> element inserts a cell into a table row.

It has the following syntax with the following parameters:

object.`insertCell(`*index*`)`

Parameter	Required / Optional	Description
index	Required (see below)	Index position where cell is to be inserted (index starts at 0). Using zero sometimes inserts a new cell at the start and sometimes at the end depending on browser. Using -1 inserts after the end of the existing last row.

Note: the index parameter is required by some browsers but optional for others.

track properties:

readyState

[JavaScriptPropertyTrackReadyState]

The `readyState` property of the JavaScript DOM object corresponding to the HTML <track> element returns the current state of a track resource.

track

[JavaScriptPropertyTrackTrack]

The `track` property of the JavaScript DOM object corresponding to the HTML <track> element returns a TextTrack object representing the text track data of the track element.

video properties (other than media properties):

videoTracks

[JavaScriptPropertyVideoVideoTracks]

The `videoTracks` property of the JavaScript DOM object corresponding to the HTML <video> element returns a VideoTrackList object indicating available video tracks.

Further Examples

[HTMLCSSJSFurtherExamples]

Many examples that illustrate specific features of HTML, CSS and JavaScript are illustrated in the Nematrian website's HTML / CSS / JavaScript Tutorial on pages that explain the specific features.

Some other, typically more sophisticated, examples included in the Tutorial are set out below:

- Drawing complex 3d spinning shapes (the example combines a spinning tetrahedron, cube, octahedron, dodecahedron and icosahedron

Animation of Spinning 3d Regular Polyhedra

[HTMLCSSJSExampleRegularPolyhedra]

In this page we illustrate how an animation involving spinning polyhedra can be created using HTML, CSS and JavaScript, and we explain the coding involved, which is set out below.

HTML

The first and last few lines involve HTML and create a canvas element, onto which the spinning polyhedra will be drawn. The canvas element is itself contained in a hyperlink, as the spinning polyhedra are mainly used in the Nematrian website to point to its HTML, CSS and JavaScript Tutorial.

Start of Script

The first part of the script, links the canvas element to a variable named `x2`, sets the canvas element size and creates a context applicable the canvas. Although canvas contexts that render 3d images directly do exist, we use here the more widely supported 2d context and include our own algorithms for projecting 3d objects onto a 2d plane.

The next part of the script defines a series of opening parameters, such as which shapes we will draw, in what order, their colouring, where they are positioned on the canvas, how big they are, how fast each is rotating, around what axis and their initial angle through which they have been rotated from the specimen layouts defined later in the code. In the colouring, the "rgba" define the red / green / blue colours of the faces and edges and the extent to which they will be transparent.

We also set some shape independent parameters, e.g. the overall x and y axes of the plane onto which we will project the resulting images and the frame rate specifying how frequently we redraw the picture on the canvas.

The shapes used are the five regular polyhedra known from antiquity. Each shape is defined by the cartesian coordinates of its vertices and by arrays that then indicate for each edge which two vertices its joins and for each face which vertices form the perimeter of the face. The initial orientation of the shape is set in a manner that simplifies the exact locations of the vertices relative to the origin. The shapes are embedded in the functions named `regularTetrahedron`, …

The script then executes the function `animate3dShapeOnCanvas`, which contains all the necessary elements to create the moving images.

Script functions

The function `animate3dShapeOnCanvas` identifies the number of shapes being drawn and initialises (in `dynamicRotAngles`) the angles to which each shape has been rotated around its axis when the animation starts. It draws the initial configuration of the shapes on the canvas by calling the `projectionIncrement` function which itself clears the canvas, resets the line path being drawn on the canvas and then projects each shape in turn onto the canvas, using a generic `project3dShapeOnCanvas` function which is designed to handle any suitably specified shape. It also updates the `dynamicRotAngles` so that the next time the shape is plotted it will be rotated slightly. The `animate3dShapeOnCanvas` function then uses the `setInterval` method to set the animation running. The `setInterval` tells JavaScript to repeatedly call a specified function (here the same `projectionIncrement` function that was used to draw the shapes initially) every set number of milliseconds (the number here being the frame rate defined earlier).

The remainder of the code is more mathematical in nature and is designed to support the `project3dShapeOnCanvas` function. For example, it includes some generic vector and matrix functions designed to facilitate rotating a shape and then projecting it onto a plane.